

Martin-Luther-Universität Halle-Wittenberg
Naturwissenschaftliche Fakultät III
Institut für Informatik

Bachelorarbeit

Sommersemester 2020

Erstgutachter: PD Dr. Alexander Hinneburg
Zweitgutachter: BA Taimur Khan

Klassifizierung von okulomotorischen Ereignissen (Fixierungen und Sakkaden) in webcam-basierten Eye-Tracking Daten

Daniel Schreiber

Inhaltsverzeichnis

1 Einleitung	3
2 Verwandte Arbeiten	6
3 Klassifikation von WebGazer-Daten in Fixationen und Sakkaden	7
3.1 Methode	8
3.1.1 Ablauf	8
3.1.2 Klassifizierung	9
3.1.3 Zielkreise	9
3.1.4 Welche Daten werden gespeichert?	10
3.2 Datensatz	11
3.3 Manuelle Nachbearbeitung des Datensatzes	17
3.4 Klassifizierung mit Algorithmen	18
3.4.1 Algorithmus: Distanz	19
3.4.2 Algorithmus: Geschwindigkeit	23
3.4.3 Algorithmus: Vektor	25
3.5 Zwischenergebnis	27
4 Erstellen eines klassifizierten WebGazer-Datensatzes	29
4.1 Methode	29
4.1.1 Inhalt	29
4.1.2 Aufbau	34
4.1.3 Angaben zur Person	35
4.1.4 Kalibrierung	36
4.1.5 Studie	39
4.2 Daten	40
4.2.1 Allgemeine Angaben	40
4.2.2 Daten von WebGazer	41
4.2.3 Daten von Tobii-Pro	42
4.2.4 Vorverarbeitung	46
4.3 Synchronisierung der Daten von WebGazer und Tobii-Pro	49
4.4 Zwischenergebnis	53
5 Diskussion	55
6 Literaturverzeichnis	59
7 Anhang	61

Gender-Hinweis: Die weibliche Form ist der männlichen Form in dieser Arbeit gleichgestellt; lediglich aus Gründen der Vereinfachung wurde die männliche Form gewählt.

1 Einleitung

Eyetracking ist eine Technologie, mithilfe der sich verfolgen lässt, was jemand mit seinen Augen beobachtet. Es braucht keine Begründung, dass sich aus diesen Informationen wichtige Rückschlüsse ziehen lassen. In der aktuellen Forschungsliteratur findet man ein breites Spektrum an Einsatzgebieten, die vom Eyetracking bei der Kommunikation (King u. a. 2019), im Spitzensport (Discombe und Cotterill 2015), beim Simultandolmetschen (Seubert 2019), in der Medizin (Harezlak und Kasprowski 2018) bis hin zum Einfluss des Blickverhaltens auf die Urteilsbildung über Politiker (Sülfow 2020) reicht. Viele dieser Studien werden mit spezieller Eyetracking-Hardware in dafür eingerichteten Laboren erstellt. Für Probanden bedeutet das einen erheblichen Aufwand, denn sie müssen den Weg ins Labor zurücklegen und sie sind zeitlich an den Studienablauf gebunden. Forschungseinrichtungen brauchen für Eyetracking-Studien viele Ressourcen. Neben der Hardware und den Räumlichkeiten muss Personal abgestellt werden, welches die Hardware bedienen und die Studie begleiten kann.

Der Ansatz des Webcam-basierten Eyetrackings beseitigt den Aufwand sowohl für die Studententeilnehmer als auch für die Forschungseinrichtungen. Die Webcam, welche heute standardmäßig in jedem Laptop, Tablet und Smartphone verbaut ist, ersetzt die spezielle Eyetracking-Hardware. Probanden führen die Studien zeitlich flexibel am heimischen Computer durch.

Forschungsprogramme können von Einrichtungen erstellt werden, die kein Eyetracking-Labor besitzen. Diese Öffnung der Eyetracking-Technologie für eine breitere Anwenderschicht, bezeichnet Alexandra Papoutsaki im Titel ihrer Dissertation als *Democratizing Eye Tracking* (Papoutsaki 2018). Auf ihre Arbeit an der JavaScript-Bibliothek *WebGazer*, die Webcam-basiertes Eyetracking im Browser ermöglicht, kommen wir weiter unten zurück.

Das Startup *Adsata* aus Halle(Saale) entwickelt eine Plattform, mit der sich moderne Eye-tracking-Studien unter Verwendung der Webcam erstellen lassen. Dafür nutzte Das Unternehmen zunächst den australischen Dienst *xLabs*¹. Deren Service übertrug den gesamten Videofeed der Webcam während einer Eyetracking-Session auf ihre Server, wo die Daten analysiert wurden. Das konnte mehrere Stunden dauern. Anschließend stellten sie die Ergebnisse über ein Web-Schnittstelle zur Verfügung. Da dieser Dienst aufwendig und teuer ist, suchte das Unternehmen nach Alternativen. Dabei fanden sie die frei-zugängliche JavaScript-Bibliothek *WebGazer*.

Im Gegensatz zu der Technologie von *xLabs* läuft *WebGazer* clientseitig im Browser. Die JavaScript-Bibliothek kann in jede Internetseite eingebunden werden und ist leicht zu verwenden. Über einen Event-Listener erhält man einen Fluss von Daten, in dem jeder Datenpunkt einen Zeitstempel (die vergangene Zeit seit WebGazer gestartet wurde) und eine (x, y) Koordinate hat, welche eine aus dem Videofeed der Webcam berechnete Angabe ist, wo der Nutzer (vermutlich) hinschaut. Adsata erstellt aus diesen Daten dann bspw. eine Heatmap, um die Ergebnisse einer Studie zu präsentieren².

Die Qualität der Daten ist der größte Nachteil von Webcam-basiertem Eyetracking gegenüber traditionellen Ansätzen. Viele Faktoren lassen sich während einer Session nicht kontrollieren, weil verschiedene Probanden unterschiedliche Webcams, Computer, Betriebssystem, Lichtverhältnisse und Verhaltensweisen haben. Zudem sind Webcams nicht auf Eyetracking spezialisiert, was sich in einer geringeren Samplingrate und verrauschteren Daten zeigt.

¹xlabsgaze.com, abgerufen am 25.11.2020.

²Eine erste Demoversion wurde am 23.11.2020 online veröffentlicht und wird gerade von verschiedenen Partnern getestet: <https://demo.adsata.com>, online abgerufen am: 25.11.2020.)

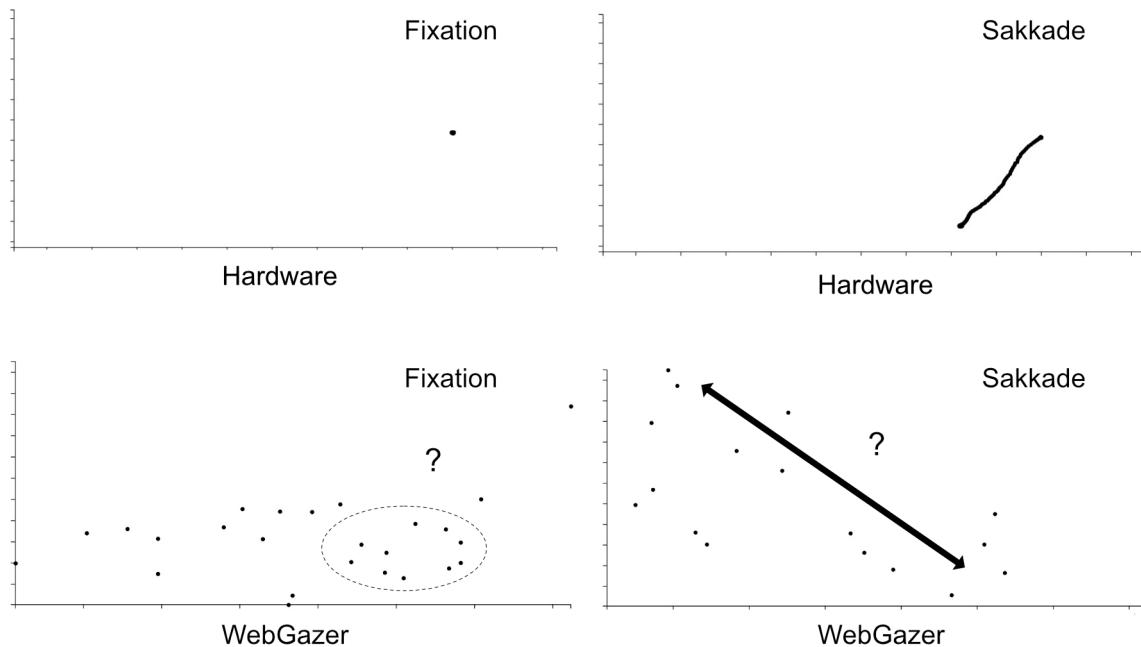


Abbildung 1: Sakkaden und Fixierungen im Vergleich. Oben ist der Datensatz abgebildet, der von Zemblys et. al zur Verfügung gestellt wird. Unten sind Daten von einer beispielhaften Webgazer-Aufnahme abgebildet, die dem Autor zur Beginn der Bachelorarbeit vorlagen. Auf der linken Seite befindet sich jeweils eine Fixierung, auf der rechten Seite eine Sakkade.

Die geringere Qualität der Daten schlägt sich in der Analyse der Daten nieder. Während professionelle Eyetracking-Hardware eine (abgestimmte) Analysesoftware mitliefern, ist die Analyse der WebGazer-Daten eine wesentliche Aufgabe von Adsata. Die basale Unterscheidung der Daten in die Ereignisse *Fixation* und *Sakkade* ist die Basis für tiefergehende Analysen. Eine Sakkade ist die Bewegung der Augen zu einem neuen Fixierungspunkt. Sie kann nicht unterbrochen oder in der Richtung geändert werden. Während einer Sakkade nimmt das Auge keine Informationen auf (Holmqvist 2011, S. 23). Die Fixation hingegen ist das Betrachten eines Ausschnittes, bei dem Informationen verarbeitet werden oder auf den zumindest die Aufmerksamkeit fokussiert ist (Holmqvist 2011, 21 f.). Fixation und Sakkaden beschreiben so wesentlich die Dynamik der Augenbewegungen und geben Aufschluss darüber, wann und in welcher Reihenfolge etwas angeschaut wurde.³

Die Schwierigkeiten, die sich bei der Analyse von Webcam-basierten Eyetrackingdaten ergeben, sind in Abbildung 1 verdeutlicht. Dort sind exemplarisch jeweils eine Fixation (links) und eine Sakkade (rechts) mit professioneller Eyetracking-Hardware (oben) und WebGazer (unten) dargestellt. Die Hardware- und WebGazer-Daten haben hier nichts miteinander zu tun. Sie sollen nur die schwierige Datenlage von Webcam-basierten Ansätzen veranschaulichen. Bei

³Es sei angemerkt, dass ich in der Fachliteratur noch auf die Ereignisse *post-saccadic oscillation*, *smooth pursuit* gestoßen bin, die aber in der folgenden Arbeit keine Rolle spielen werden.

den Daten der Hardware sind Fixation und Sakkade nicht nur deutlich zu unterscheiden, sondern auch die Dichte der Daten ist bemerkenswert. Trotz der hohen Anzahl an Datenpunkten gibt es keinen einzigen Ausreißer. Die Fixation konzentriert sich auf ein einzige Stelle und die Sakkade ist vollständig zusammenhängend und detailreich. Demgegenüber lässt sich eine Fixation bzw. Sakkade in den WebGazer-Daten nur vermuten oder gar zurückweisen.

Die Klassifikation der Daten in Fixation und Sakkaden erzeugt WebGazer nicht.⁴ Die geringe Samplingrate und die breite Streuung machen sie zu einem Problem. Um diese Aufgabe zu lösen, suchte Adsata nach neuen Ansätzen, die mit dieser Datenlage umgehen können. Dabei stießen sie auf den wissenschaftlichen Artikel von Zemblys et. al., in welchem die Autoren beschreiben, wie sie einen *Random Forrest* Klassifizierer mit Hardwaredaten wie in Abbildung 1 trainiert und getestet haben (Zemblys u. a. 2018). Die Autoren kommen zu dem Schluss:

that machine-learning techniques lead to superior detection compared to current state-of-the-art event detection algorithms and can reach the performance of manual coding.

Diese Methode hat zwei Vorteile:

1. Es müssen keine Schwellenwerte für Algorithmen gesetzt werden (Zemblys u. a. 2018, S. 2).
2. Das Modell funktioniert auch auf Daten mit geringer Samplingrate, die ein hohes Rauschen aufweisen.

Der erste Punkt erlaubt das Klassifizieren von Daten, ohne die Kenntnisse von Experten, weil keine weiteren Einstellungen vorzunehmen sind. Der zweite Punkt passt gut zu den oben beschriebenen WebGazer-Daten. Er verspricht, trotz schwieriger Datenlage, eine sinnvolle Klassifikation, die andere (klassische) Algorithmen nicht leisten:

Most of these algorithms work well within the assumptions they make of the data. Examples of common assumptions are that the input must be high-quality data, or data recorded at high sampling frequencies [...]. All algorithms come with overt settings that users must experiment with to achieve satisfactory event detection in their data set, or covert settings that users have no access to. When the sampling frequency is too low, or too high, or the precision of the data is poor, or there is data loss, many of these algorithms fail (Holmqvist et al. 2012, 2016).

Das Unternehmen Adsata ist daran interessiert, ein solches Modell zu entwickeln, zu testen und für ihre Plattform zu verwenden. Die vorliegende Bachelorarbeit widmet sich der Annäherung an dieses Ziel. Es wird zum einen der Frage nachgegangen, ob eine Klassifikation von WebGazer-Daten in Fixation und Sakkaden sinnvoll ist. Dafür wird ein Datensatz erstellt, bei dem Nutzer festgelegte Augenbewegungen ausführen. Mithilfe visueller Darstellungen und einfacher Algorithmen wird dann gezeigt, dass sich Datengruppen als Sakkaden und Fixationen unterscheiden lassen. Zum anderen löst die Bachelorarbeit das Problem, dass zur Zeit kein

⁴Es gibt ein Github-Issue, welches fehlende Algorithmen für Fixationen und Sakkaden als mögliches Feature angibt. Das Issue wurde 2016 eröffnet und ist bisher nicht bearbeitet worden. <https://github.com/brownhci/WebGazer/issues/16>, abgerufen am 27.11.2020.

gelabelter Datensatz vorhanden ist, der zum Trainieren eines Klassifizierers benötigt wird. Dazu wurde eine Methode entwickelt, Daten eines Webcam-basierten und Hardware-basierten Eyetrackingsystems gleichzeitig aufzunehmen und die Label der Hardwaredaten auf die von WebGazer zu übertragen. Die Plausibilität der so erzeugten Daten muss geprüft und das Ergebnis diskutiert werden. In einem Ausblick beschreibt die Arbeit das Zwischenergebnis, welches auf dem Weg zu einem Ansatz mit maschinellem Lernen erreicht wurde und stellt die Schritte dar, die weiterhin zu unternehmen wären.

2 Verwandte Arbeiten

Die vorliegende Bachelorarbeit stützt sich auf die Webcam-Eyetracking-Technologie von WebGazer⁵. Diese ist im Rahmen von Forschungen an der Brown University entstanden. Zu WebGazer sind mehrere Publikationen erschienen. Ausführlich vorgestellt wird die Technologie in der zugehörigen Doktorarbeit von Papoutsaki (Papoutsaki 2018, Kapitel 3). Eine komprimierte Form des Inhalts wurde 2016 bereits in dem Artikel *WebGazer: Scalable Webcam Eye Tracking Using User Interactions* (Papoutsaki, Sangkloy u. a. 2016) vorgestellt. Papoutsaki et. al. haben einen Datensatz mit 51 Probanden aufgenommen, bei dem dieselbe Eyetracking-Hardware (Tobii Pro X3-120) verwendet wurde, wie in Kapitel 4 der vorliegenden Arbeit (Papoutsaki, Gokaslan u. a. 2018), (Papoutsaki 2018, Kapitel 5). Zusätzlich wurden über die Webcam der gesamte Videofeed im MP4-Format gespeichert. Die Zusammenführung der Daten muss von Forschern selber übernommen werden⁶ und ist nur mit einer veralteten Version von WebGazer möglich. Der Datensatz wurde erstellt, um es Forschern zu erlauben, Webcam-basierte und Hardware-basierte Eyetracking-Technologien zu vergleichen. Die Hardwaredaten liefern keine Klassifikation in Sakkaden und Fixationen mit. Im Unterschied dazu nutzt die vorliegende Bachelorarbeit in Kapitel 4 die aktuelle Version von WebGazer und Tobii Pro X3-120, um Daten gleichzeitig aufzunehmen und die Label von der Hardware auf die Daten von WebGazer zu übertragen.

Die Entwickler von WebGazer haben ihre Technologie unter dem Namen SearchGazer für den Einsatz bei der Websuche weiterentwickelt und diese mit Probanden getestet (Papoutsaki, Laskey und Huang 2017). Sie konnten im Allgemeinen die Ergebnisse bisheriger Forschungen mit Eyetracking-Hardware zur Websuche reproduzieren. Unterschiede in den Ergebnissen führten die Autoren unter anderem auf fehlende Algorithmen für Fixationen und Sakkaden zurück, die sie in zukünftigen Arbeiten integrieren wollen (Papoutsaki, Laskey und Huang 2017, S. 25).

Ob WebGazer eine Alternative zu Eyetracking-Hardware ist, wurde auch in der Kognitivpsychologie erforscht (Semmelmann und Weigelt 2018). Die Autoren untersuchten mit Probanden drei unterschiedliche Aufgaben und kommen zu dem Schluss, dass Eebcam-basiertes Eyetracking für diese drei Aufgaben eine geeignete Alternative darstellt. Zudem drückten die Autoren ihre Überzeugung aus, dass die Hard- und Software im Bereich Webcam-Eyetracking sich in Zukunft weiter verbessern wird und sich damit noch weitere Untersuchungsmöglichkeiten eröffnen.

Eine Alternative zu WebGazer bietet TurkerGaze (Xu u. a. 2015). Die Entwickler haben, um

⁵<https://webgazer.cs.brown.edu/>, abgerufen am 20.12.2020

⁶Eine Beschreibung findet sich online unter: <https://webgazer.cs.brown.edu/data/>, abgerufen am 19.12.2020

ihren Webcam-basierten Ansatz zu evaluieren, ebenfalls einen Datensatz gleichzeitig mit einem Hardwaresystem⁷ aufgenommen. Um das Klassifizieren der Daten mithilfe der Hardware ging es nicht. TurkerGaze wird seit 5 Jahren nicht mehr weiterentwickelt.⁸.

Wie in der Einleitung beschrieben ist der Ausgangspunkt dieser Arbeit ein Artikel von Zembly et. al. (Zemblys u. a. 2018). In diesem wird ein *Random Forest* Klassifizierer vorgestellt und evaluiert. Dafür wurde ein Datensatz mit dem EyeLink 1000 System⁹ erstellt, welcher manuell von Zemblys nachbearbeitet wurde (Zemblys u. a. 2018, Baseline dataset). Der Datensatz wurde dann durch additives weißes gaußsches Rauschen vergrößert, um verrauschte Daten mit geringerer Samplingrate zu imitieren (Zemblys u. a. 2018, Data augmentation). Der Ansatz wurde mit einem weiteren Datensatz repliziert (Raimondas, C und Kenneth 2018), um die Ergebnisse zu verifizieren.

Einen vergleichenden Überblick über bisherige Klassifikationsalgorithmen bietet die Arbeit von Andersson et. al. (Andersson u. a. 2016). Sie stellt 10 Algorithmen vor und testet die Güte ihrer Klassifikationen anhand eines Datensatzes, welches mit dem SMI HiSpeed 1250 System¹⁰ aufgenommen wurde. Dieser Datensatz wurde dann von zwei Experten unabhängig manuell klassifiziert, so dass zwei Datensätze zum Testen zur Verfügung standen.

Eine Übersicht über Datensätze, die durch Vernwendung von Eyetracking-Hardware entstanden sind, bietet die Arbeit von Winkler und Subramanian (Winkler und Subramanian 2013). Die Datensätze können zum Vergleich von Klassifikationsalgorithmen herangezogen werden, enthalten aber keine Eyetracking-Daten von Webcam-basierten Ansätzen.

In dieser Arbeit werden zwei Datensätze mit unterschiedlichen Methoden erzeugt (vgl. Kapitel 3 und 4). Gemeinsam ist ihnen aber, dass die Experimente unter kontrollierten (Labor-)Bedingungen ablaufen. Die zweite Studie wird später noch durch Daten aus Sessions erweitert, die Probanden von zu Hause durchführen können. Dass die Datenqualität nicht abnehmen muss, weil Teilnehmer nicht die gleiche Motivation und den gleichen Fokus wie im Labor haben, beschreibt eine Forschungsarbeit von Germine et. al. (Germine2012).

3 Klassifikation von WebGazer-Daten in Fixationen und Sakkaden

Das erste Kapitel beantwortet die Frage, ob sich WebGazer-Daten sinnvoll in Fixationen und Sakkaden klassifizieren lassen. Um das zu tun, wird ein Datensatz mit WebGazer erstellt. Die verwendete Methode (Kapitel 2.1) zur Erstellung, verlangt vom Nutzer Handlungen, die erwartbare Fixations- und Sakkadengruppen erzeugen und es über eine Heuristik erlaubt, die Daten ihren Gruppen entsprechend zu bezeichnen. In einer Analyse des Datensatzes (Kapitel 2.2) finden sich diese Gruppierungen wieder. Um problematische Bezeichnungen einzelner Datenpunkte zu beheben, die sich unweigerlich aus der Heuristik ergeben, wird der Datensatz manuell nachbearbeitet (Kapitel 2.3). Einfache Algorithmen zeigen weiter, dass sich die Gruppen nicht nur visuell sondern auch mathematisch in den Daten finden lassen (Kapitel

⁷Eyelink 1000, <https://www.sr-research.com/eyelink-1000-plus/>, abgerufen am 19.12.2020

⁸<https://github.com/PrincetonVision/TurkerGaze>

⁹<https://www.sr-research.com/eyelink-1000-plus/>, abgerufen am 19.12.2020

¹⁰<https://www.inition.co.uk/product/sensomotoric-instruments-iview-x-hi-speed/>, abgerufen am 19.12.2020

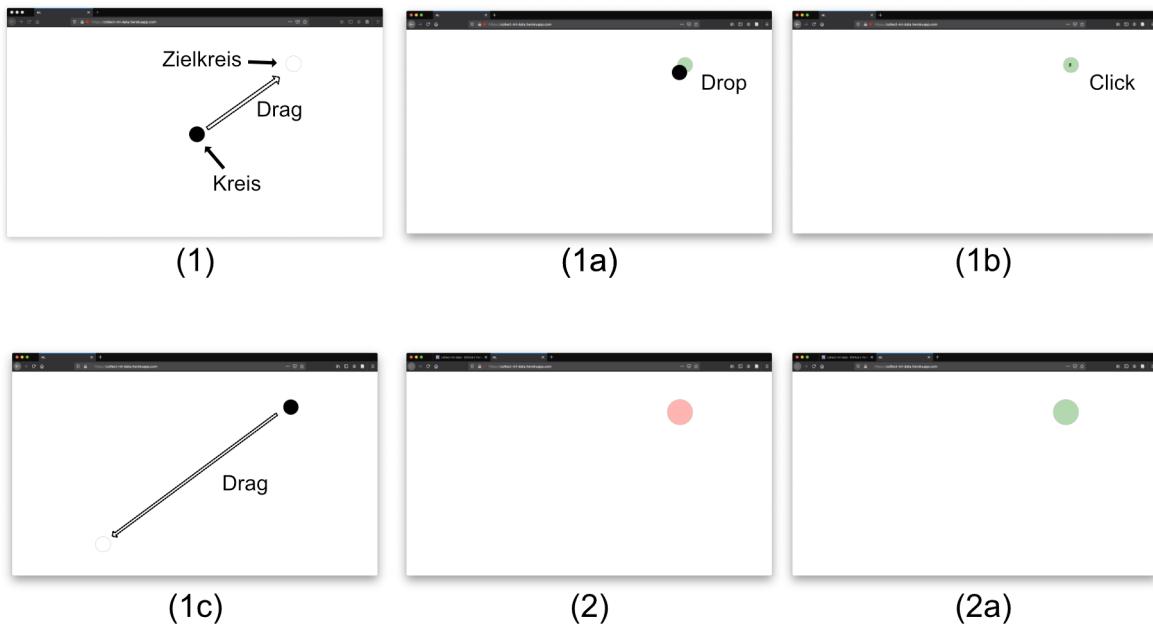


Abbildung 2: Methode der ersten Datenerhebung. Das Verfahren lässt sich in zwei Teile gliedern: das Verschieben und Klicken von Kreisen mit der Maus (1) und das bloße Anschauen eines Kreises (2)

2.4). Die Schlussfolgerung ist, dass sich die Daten von WebGazer sinnvoll als Fixationen und Sakkaden unterscheiden lassen (Kapitel 2.5).

3.1 Methode

Die Methodik in diesem Kapitel schränkt die Interaktion des Nutzers mit dem Computer ein, um WebGazer-Daten zu erhalten, die in einer kontrollierten Umgebung entstehen. Dabei ist das (mehrfache) Klicken mit der Maus auf eine Stelle der Indikator für eine Fixation. Über eine Drag-and-Drop-Geste mit der Maus soll eine Sakkade induziert werden. Um die Daten mit einem größeren Freiheitsgrad vergleichen zu können, gibt es noch einen zweiten Teil, welcher auf ein Eingabegerät verzichtet und nur die Augenbewegungen als Eingabe nutzt.

3.1.1 Ablauf

Das Verfahren der Datenerhebung lässt sich in zwei Teile unterteilen. Im ersten Teil muss ein Kreis mithilfe der Maus in einen Zielkreis verschoben werden (vgl. Abb. 2, (1, 1a, 1c)). Dazu klickt man auf den Kreis, um ihn dann mit gehaltener Maustaste zu verschieben (*drag*). Erreicht man mit dem Kreis den Zielkreis, färbt dieser sich grün. Das zeigt an, dass die gehaltene Maustaste losgelassen werden kann, um den Kreis im Zielkreis zu positionieren

(*drop*). Die Kreis muss dabei nicht genau auf dem Zielkreis ausgerichtet werden, es reicht schon, ihn zu berühren. Diesen Ablauf bezeichne ich als *Drag-and-Drop-Phase*.

Ist der Kreis im Zielkreis platziert (1b), dann bleibt der Kreis grün und in der Kreismitte erscheint die Zahl fünf. Nun muss der Kreis fünfmal angeklickt werden, wobei sich jedes mal die Zahl um eins verringert. Diese Interaktion bezeichne ich als *Click-Phase*. Nach dem letzten Klick wird der Kreis wieder schwarz und es erscheint ein neuer Zielkreis (1c). Damit befindet man sich wieder im *Drag-and-Drop*-Ablauf.

Das ganze wiederholt sich mit insgesamt 28 Zielkreisen (vgl. Abb. 3), aufgeteilt in zwei Durchläufe mit je 14 Punkten. Im Ablauf ist die Aufteilung nicht bemerkbar, aber es werden zwei getrennte Datenobjekte erzeugt. Dabei dient der erste Durchlauf der Kalibrierung von WebGazer und der zweite Durchlauf wird später für die Analyse verwendet.

Ohne einen expliziten Übergang schließt sich der zweite Teil an. Dieser besteht aus fünf Kreisen die jeweils in den Ecken des Bildschirms und in der Mitte platziert sind. Diese müssen nun weder verschoben noch angeklickt, sondern lediglich angeschaut werden. Liegt ein Datenpunkt von WebGazer in dem Kreis, dann wechselt er die Farbe von rot über gelb bis grün (vgl. Abb. 2, (2)). Anschließend erscheint der Kreis an der nächsten Position. Es sind also drei Datenpunkte von WebGazer auf einem Kreis nötig, um im Verfahren weiterzukommen. Die Reduzierung der Punkte auf fünf ist nötig, weil es bei diesem Verfahren sein kann, dass das Ziel nicht ordentlich anvisiert wird, was die ganze Prozedur langwierig macht, weil es keine andere Möglichkeit gibt, den Prozess zu beenden. Erst nachdem alle fünf Kreise erfolgreich angeschaut wurden, ist die Datenerhebung beendet.¹¹

3.1.2 Klassifizierung

Der oben beschriebene Ablauf erlaubt folgende Heuristik, um die Datenpunkte von WebGazer beim Sammeln zu klassifizieren. Im ersten Teil werden alle Datenpunkte die in den *Drag-and-Drop*-Phasen gesammelt werden, als *saccade* bezeichnet. Solange also der Kreis mit der Maus verschoben wird, bewegt sich das Auge (mit Mauszeiger und Kreis) zum Zielkreis. Nachdem der Kreis losgelassen wurde und die *Click-Phase* beginnt, schaut das Auge auf den anzuklickenden Kreise, weshalb das Label *fixation* vergeben wird. Aus diesem Ablauf werden also Annahmen über die Augenereignisse getroffen. Diese sind zwar plausible (Huang, White und Buscher 2012), müssen aber nicht stimmen.

Im zweiten Teil ist die Annahme noch etwas unschärfer, weil nun keine Mausereignisse mehr für die Entscheidung hinzugezogen werden können. Hier werden alle Datenpunkte als *fixation* gelabelt, die zwischen dem ersten und letzten Datentreffer des Kreises liegen. Alle anderen Daten werden als *saccade* bezeichnet.

3.1.3 Zielkreise

Es ist zu vermuten, dass die Daten an den Rändern des Bildschirms unsauberer sind (Semmelmann und Weigelt 2018, S. 462). Deshalb werden für einen ersten Datensatz die Zielpunkte so gewählt, das links und rechts jeweils 20% Rand und oben und unten jeweils 10% Rand

¹¹Das ist zum Beispiel ein Problem, wenn man den Radius des Kreises sehr klein wählt. In diesem Fall kann es passieren, dass die Datenerhebung abgebrochen werden muss, weil die Kreise durch den Blick einfach nicht getroffen werden. Nach einem Experimentieren wurde der Radius des Kreises auf 30 Pixel festgelegt, was in fast allen Fällen zu einer recht schnellen Abfertigung der Kreise führte.

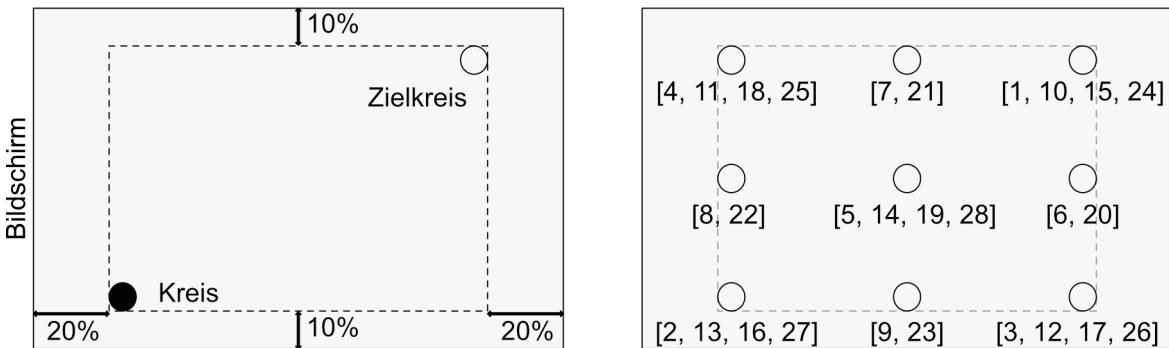


Abbildung 3: Die linke Abbildung zeigt den Aufbau und die Maße für die Datenerhebung. Die rechte Abbildung zeigt alle Zielkreise. Die Zahlen in den eckigen Klammern geben die Reihenfolge der Zielkreise wieder.

vom Browser-Fenster sind (vgl. Abb. 3). Innerhalb dieses so zentrierten Rechtecks werden die Zielkreise mit einem Radius von 30 Pixel an dessen Ränder sowie mittig platziert.

Die Situation bei der Datenaufnahme kann für verschiedene Anwender unterschiedlich sein. Um die Darstellung über verschiedene Bildschirm- und Browserfenstergrößen zu vereinheitlichen, sind alle Angaben in Prozent. Dabei kommt die zusätzliche Annahme hinzu, dass das Verhältnis von Bildschirmgröße und Sitzentfernung gleich ist. Wer also einen größeren Bildschirm hat, der sitzt davon weiter entfernt, als wenn er einen kleineren Bildschirm hätte. So dass die Entfernungungen für das Blickverhalten zwischen Anwendern ungefähr identisch ist. Mit Hilfe dieser Annahme lassen sich die Daten später vergleichen und gemeinsam auswerten, weil ein WebGazer-Datenpunkt bei der Koordinate (20 %, 10 %) dann über alle Anwender hinweg immer die gleiche Aussage beschreibt. Die WebGazer-Koordinaten sind zwar in absoluten Angaben, können aber mit der zusätzlich gespeicherten Browserfenstergröße normalisiert werden. Da die absoluten Positionen der Zielkreise aber unterschiedlich sein können, werden diese in einem zusätzlichen Datenobjekt gespeichert (vgl. unten). Für den Ablauf bedeutet das allerdings, dass die Größe des Browserfensters während einer Session nicht verändert werden darf. Allgemein sollte der Browser im Vollbildschirm-Modus verwendet werden.

3.1.4 Welche Daten werden gespeichert?

Neben einigen Metadaten werden drei Listenobjekte erzeugt, die die Zeitstempel und die (x,y) Koordinaten von WebGazer sowie das erzeugte Label enthalten. In der ersten Liste sind Kalibrierungsdaten, die für weitere Berechnungen nicht verwendet werden. Die zweite Liste enthält dann die eigentlichen Daten vom *Drag-and-Drop/Click*-Verfahren, während die dritte Liste die Daten vom Blickverfahren enthält (vgl. Abb. 4).

Für die Berechnungen sind einige Metadaten nützlich. Um verschiedene Sessions unterscheiden zu können, wird der Zeitstempel vom Zeitpunkt der Übertragung der Daten in die Datenbank gespeichert. Zusätzlich wird der Name des Anwenders gespeichert. Dieser kann vorher vom Anwender selbstständig vergeben werden. Außerdem wird danach gefragt, ob eine Brille ge-

Beispielobjekt aus der Datenbank

```

_id: ObjectId("5f4e0828a6ba17062570a2bd")
timestamp: 1598949413945
name: "Daniel"
glasses: "true"
browser: "Netscape"
window.innerWidth: 1440
window.innerHeight: 803
marginHeight: 80.30000000000001
marginWidth: 288
> calibrationTargets: Array
> clickTargets: Array
> gazeTargets: Array
> calibrationData: Array
> clickData: Array
> gazeData: Array

```

Array von Koordinaten der Zielkreise
x : Float
y : Float

Array von WebGazer-Daten mit Label
timestamp : Float
x : Float
y : Float
label : String ("fixation" | "saccade")

Abbildung 4: So werden die Daten in MongoDB gespeichert. Auf der linken und rechten Seite sind jeweils Beispiele für die Objektstrukturen der einzelnen Arrays.

tragen wird. Neben diesen zwei Angaben durch den Anwender werden noch Informationen über den Browser (Angabe des Browsers, Fenstergröße) und jeweils eine Liste für die Ziellkreise der einzelnen Verfahrensschritte (Kalibrierung, Drag-and-Click-Verfahren, Blickverfahren) aufgenommen.

3.2 Datensatz

In diesem Abschnitt wird ein Beispiel-Datensatz mit zwei Dokumenten vorgestellt, der nach der oben beschriebenen Methode aufgenommen wurden.¹² Er enthält:

- 1290 Datenpunkte für die Kalibrierung,
- 1255 Datenpunkte für den ersten Teil (*Drag-and-Drop/Klick*) und
- 596 Datenpunkte für den zweiten Teile (nur anschauen).

Jeder dieser Datenpunkte besteht (wie oben beschrieben) aus einem Zeitstempel (*timestamp*), dessen Nullpunkt jeweils der Start von WebGazer ist, einer (x, y) Koordinate sowie einem Label als *fixation* oder *saccade*. Die Label sind nach der oben beschriebenen Heuristik vergeben worden. Die Fixationen und Sakkaden sind ausgewogen verteilt:

¹²Während der Entwicklung sind verschiedene Datensätze mit vielen Dokumenten entstanden, auf die hier aber aus Platzgründen nicht weiter eingegangen wird.

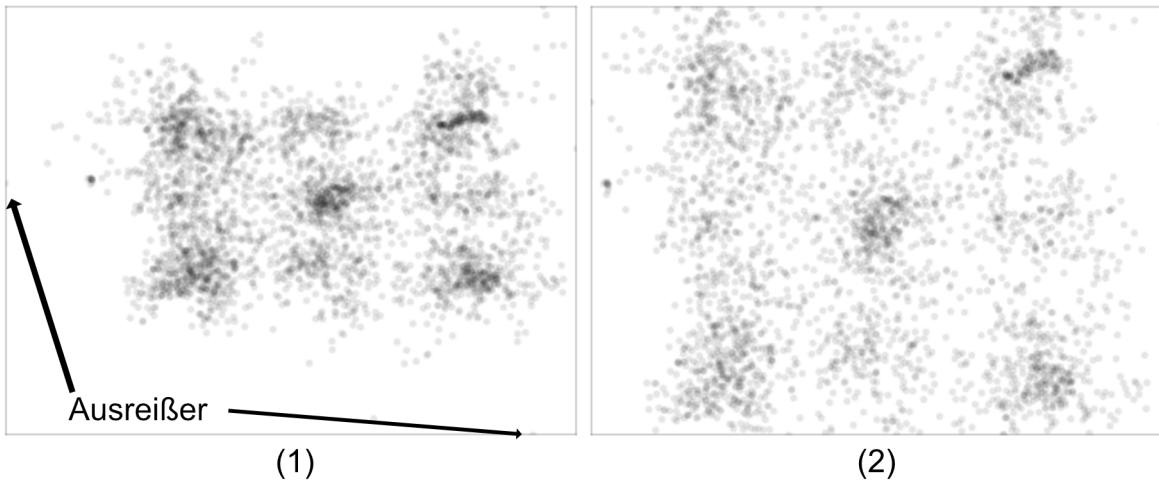


Abbildung 5: Alle Datenpunkte des Beispieldatensatzes (ausgenommen die Kalibrierungsdaten). Links ungefiltert. Rechts gefiltert anhand der Fenstergröße des Browsers. Die zusätzlichen Kreise veranschaulichen, wo sich die einzelnen Zielkreise vermuten lassen.

	Sakkaden	Fixationen
Kalibrierung	736 (57 %)	554 (43 %)
Drag-and-Drop	660 (53 %)	595 (47 %)
Blickverfahren	277 (46 %)	319 (54 %)

Diese Gleichverteilung ergibt sich aus der Tatsache, dass man fünf mal auf den Kreis klicken muss. Dadurch dauert die Klick-Phase ungefähr gleichlang wie die Drag-and-Drop-Phase. Alle Datenpunkte¹³ sind in Abbildung 5 dargestellt. Es gibt extreme Ausreißer, bei denen der Datenpunkt weit außerhalb des Browserfensters liegt (1). Wenn diese Ausreißer gefiltert werden, ist die Darstellung genauer (2), aber es wird auch schwieriger die neun Bereiche der Zielkreise zu erkennen. Abbildung 6 verstärkt durch zusätzliche Kreise die Regionen, wo sich Gruppen von Datenpunkten sammeln. Bei dieser Visualisierung sind die verschiedenen Bereiche erkennbar. Dabei fällt auf, dass es einige der neun Regionen gibt, in denen sich mehr Datenpunkte (oben links, mittig) befinden als in anderen Regionen (oben mittig, mittig rechts). Das könnte daran liegen, dass der Teilnehmer bei einigen Zielkreisen mit dem Klicken schneller war und deshalb dort weniger hingeschaut hat. Um das zu überprüfen, könnte der Datensatz vergrößert oder die aufgewendete Zeit gemessen werden. Für diese Bachelorarbeit soll es aber erst mal reichen, dass sich die neun Zielkreise in den Daten wiederfinden lassen. Stellt man die Datenpunkte der ersten Session sortiert nach ihren Labelgruppen dar, werden die neun Zielkreise und die jeweiligen Bewegungen dorthin sichtbar. Die Abbildung 7 stellt den ersten Teil (*Drag-and-Drop/Click*) der ersten Session dar. Die Zahlen zeigen die Reihen-

¹³Die Kalibrierungsdaten spielen jetzt keine Rolle mehr.

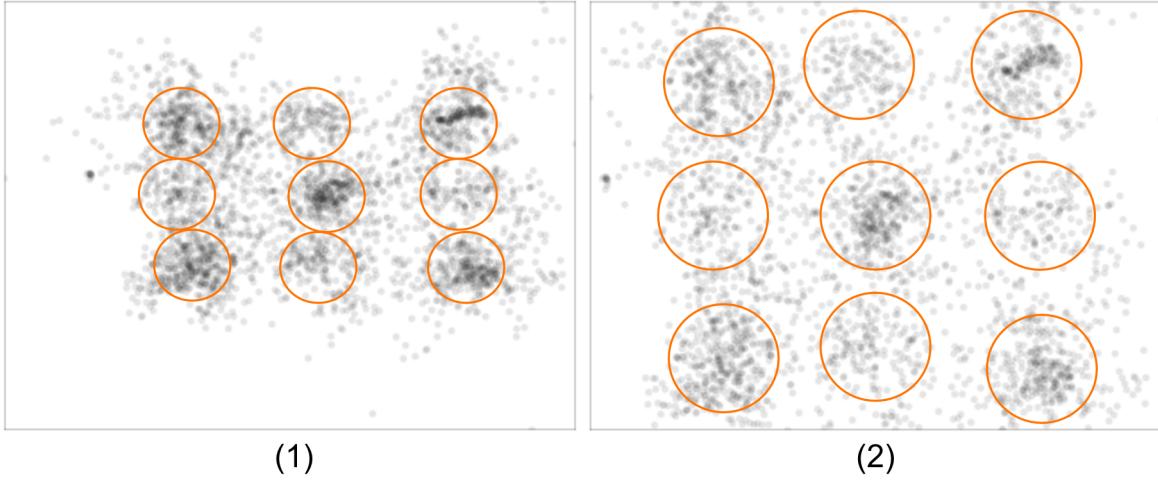


Abbildung 6: Alle Datenpunkte des Beispieldatensatzes (ausgenommen die Kalibrierungsdaten). Links ungefiltert. Rechts gefiltert anhand der Fenstergröße des Browsers.

folge der aufgenommenen Daten. Betrachtet man nur die Bilder der geraden Zahlen, dann erhält man alle Fixationen in ihrer Reihenfolge. Die ungeraden Zahlen enthalten alle Sakkadengruppen. Bei den Abbildungen der Fixationen erkennt man deutliche Gruppierungen um einen Punkt. Die einzige Ausnahme ist Nummer (12). Von diesen Fixationsgruppen heben sich die Sakkaden ab. Eine prinzipielle Unterscheidung dieser beiden Klassen lässt sich also feststellen.

Schauen wir uns die erste Session des Datensatzes etwas genauer an. Die Abbildung 8 zeigt die Qualität der Heuristik: Die Fixationsgruppen (1-3) sind als solche zu erkennen. Die meisten Datenpunkte sind richtig bezeichnet. Im Datensatz der ersten Session gibt es (von 19 Fixationen) nur zwei Ausnahmen. Bei der ersten Ausnahme sind die Datenpunkte über einen größeren Umfang verteilt (2). Auf die zweite Ausnahme komme ich weiter unten zurück. Die Klassifikation als Fixation gelingt gut, weil sie erst beginnt, wenn der Zielkreis in der *Drag-and-Drop*-Phase erreicht wurde und bereits endet, wenn der letzte Mausklick auf dieses Ziel getätig ist. Während also alle Daten als Fixation gelabelt werden, schaut der Teilnehmer auf den entsprechenden Zielkreis und das spiegelt sich in den Daten wieder. Anders verhält es sich bei den Sakkadengruppen (4-6). In diesen sind noch Datenpunkte aus der Fixation mit enthalten. Die Klassifikation als Sakkade beginnt mit dem letzten Mausklick. Es braucht dann aber noch einige Zeit, bis der neue Kreis angeklickt und verschoben wird. Die Teilabbildung (3) sieht aus wie eine sehr kompakte Fixation, aber wenn die folgenden als Sakkade klassifizierten Datenpunkte mit dazugenommen werden (3a), dann zeigt sich, dass die Fixation deutlich mehr Punkte (ca. 30 % mehr) enthält und einen größeren Umfang hat. Genauso verhält es sich beim Loslassen der Kreises im Zielkreis (4). Bei der Bewegung von oben links in die Mitte entsteht beim Loslassen bereits ein kleiner Datenhaufen. Der Teilnehmer fixiert bereits das Ziel, auch wenn er den Kreis noch nicht losgelassen hat.

Im ganzen Datensatz kommt es nur einmal vor, dass Punkte als Fixation klassifiziert werden,



Abbildung 7: Gezeigt werden die Datenpunkte der ersten Phase (*Drag-and-Drop/Click*) von Session 1 nach ihren Labelgruppen sortiert. Die Zahlen geben die Reihenfolge wieder. Liest man nur die ungeraden Zahlen, dann ergibt sich die Reihenfolge aller Sakkaden, bei den geraden Zahlen die Reihenfolge aller Fixationen.

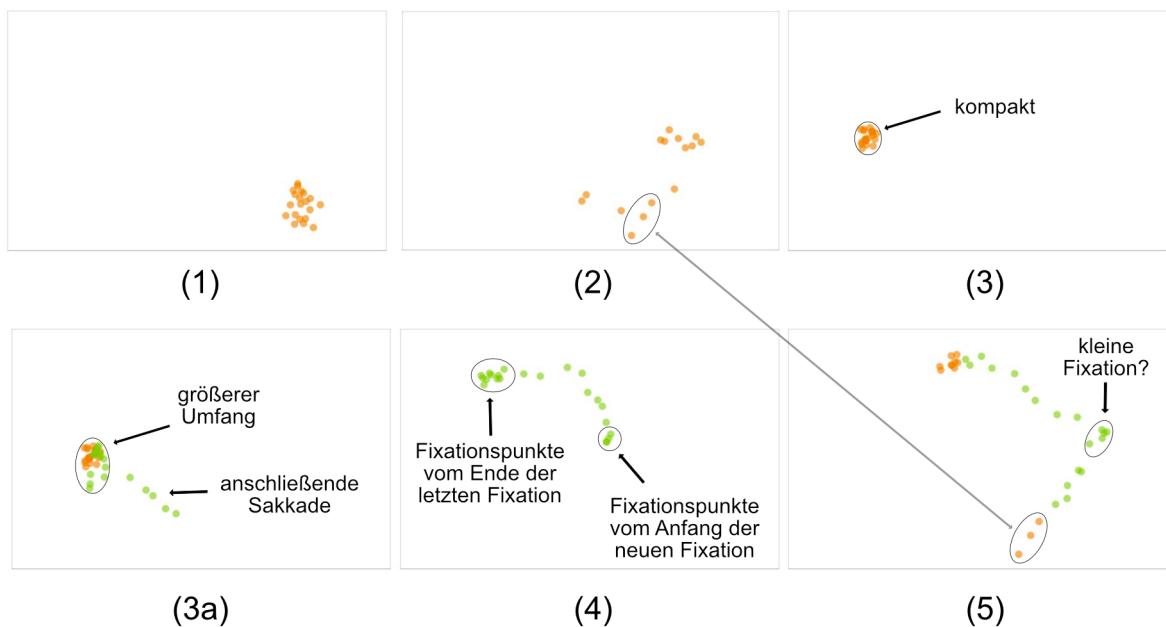


Abbildung 8: Ausgewählte Gruppen von Datenpunkten aus der ersten Session des Datensatzes (vgl. Abb. 7). An ihnen wird die Qualität der Heuristik demonstriert: Im Allgemeinen sind die Unterschiede der Label Fixation und Sakkade deutlich. Insbesondere die Fixationsgruppen sind als solche eindeutig. Hingegen der Start einer Sakkade bzw. das Ende (die Dauer) einer Fixation sind verschwommen.

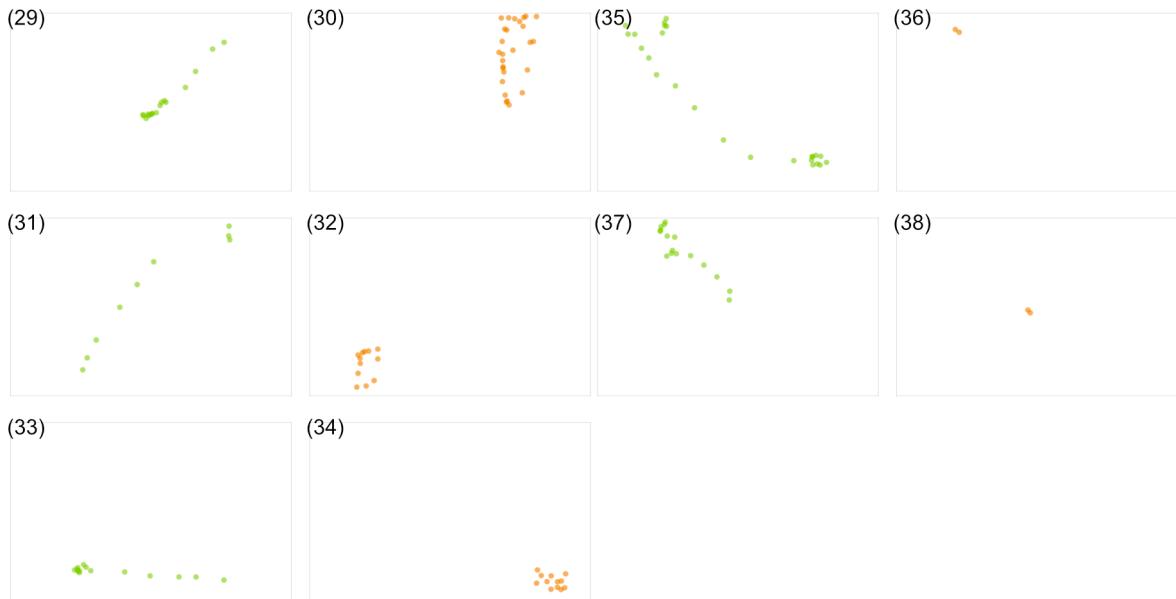


Abbildung 9: Beispiel-Datensatz - Blickverfahren. Die Abbildung zeigt alle fünf Fixationen des zweiten Teils der Datenerhebung.

obwohl sie (zumindest scheinbar) eine Sakkade sind (5). Die falsch klassifizierten Datenpunkte stammen dabei aus einer Fixationsgruppe, die starkes Rauschen enthält (2). Außerdem ist hier die Richtung der Sakkade nicht eindeutig und es stellt sich die Frage, ob es auf dem Weg zum oberen mittleren Bildschirmbereich eine kleine Fixation am rechten Bildschirmrand gibt. Werden die Daten erhoben, während der Teilnehmer nur noch in den Zielkreis schauen muss (zweiter Teil der Datenerhebung), dann zeigen sich bei den Sakkaden und den oben beschriebenen Phänomenen keine Unterschiede. Hingegen bei den Fixationen kommt ein neues Phänomen dazu (vgl. Abb. 9). Während wir eine weit gestreute Fixation (30) sowie deutlichere Fixationen (32, 34) auch schon oben hatten, kommen nun noch sehr kurze, nur aus zwei Datenpunkten bestehende Fixationen (36, 38) hinzu. Dieses Phänomen liegt an der Datenerhebungsmethode. Die Daten werden erst als Fixation klassifiziert, wenn der erste Datenpunkt den Zielkreis trifft. Da nur drei Datenpunkte benötigt werden, um zum nächsten Schritt zu gelangen, gehören dann zu einer Fixation nur zwei Datenpunkte, falls alle Koordinaten im Zielkreis liegen. Hier kommt die oben beschriebene Problematik des Kreisradius ins Spiel. Wird der Kreisradius zu klein gewählt, kann es bei diesem Teil dazu führen, dass der Prozess nicht weitergeht, weil die Datenpunkte alle außerhalb des Zielkreises (30) liegen. Wählt man allerdings den Radius zu groß, dann gibt es nur sehr wenige Datenpunkte, weil die Trefferquote des Kreises hoch ist.

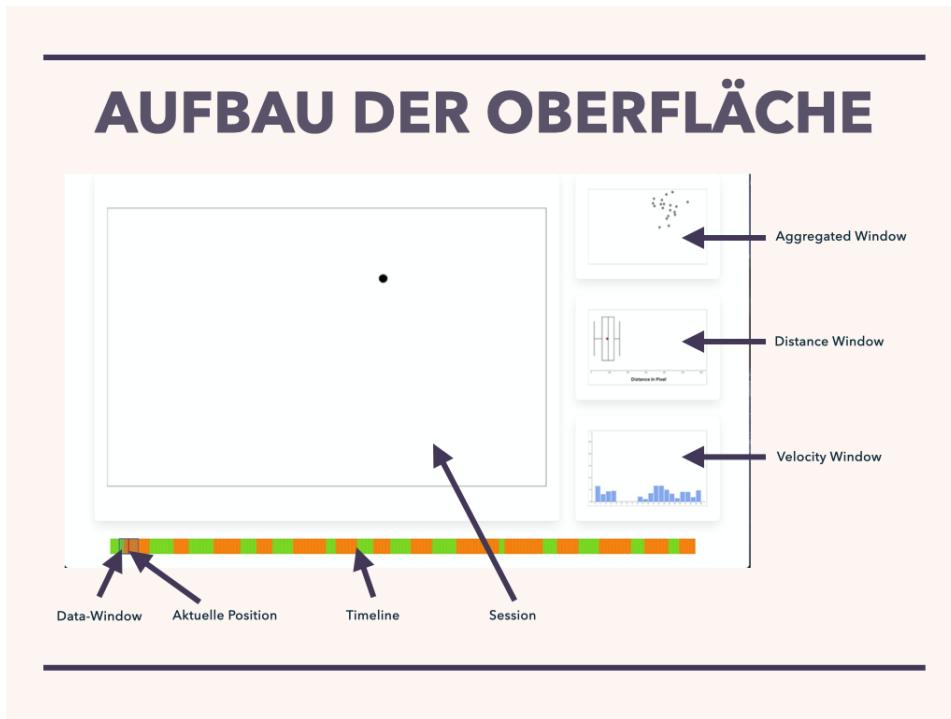


Abbildung 10: Diese Webapplikation wurde in dieser Arbeit verwendet, um Daten von WebGazer zu visualisieren und manuell zu klassifizieren.

3.3 Manuelle Nachbearbeitung des Datensatzes

Bevor Algorithmen Eyetracking-Daten klassifizierten, mussten das Experten händisch übernehmen (Zemblys u. a. 2018, S. 1). Auch heute werden Daten noch hand-gelabelt, um Datensätze zu erstellen, mit denen die Qualität von Algorithmen evaluiert werden (Zemblys u. a. 2018), (Winkler und Subramanian 2013). Wie ein Experte dabei vorgeht und welche Kriterien er anlegt, beschreibt Zemblys et. al. in einem Report (Raimondas, C und Kenneth 2018). Er stellt dort eine grafische Oberfläche vor, die Experten verwenden, um die Daten zu visualisieren und Entscheidungen für die Klassifikation zu treffen. Zemblys nutzt zwei Schritte: Zuerst lässt er Algorithmen die Daten klassifizieren, um sie anschließend manuell nachzubearbeiten. So muss der Experte nur kritische Bezeichnungen entscheiden. Dieser Ansatz wird hier übernommen. Die durch die Heuristik klassifizierten Daten sollen manuell nachkorrigiert werden, um so einen verlässlichen Datensatz von WebGazer zu erstellen. Dafür wurde eine Webapplikation geschrieben, die die Daten visualisiert und mithilfe der Tastatur die Label der Daten ändert.

Die Oberfläche der Webapplikation (vgl. Abb. 10) besteht aus fünf Teilbereichen:

1. Timeline

Die Timeline zeigt für jeden Datenpunkt in einem Datensatz einen senkrechten Strich an. Die Farbe ist orange (Fixation), grün (Sakkade) oder grau (Undefiniert). Ein roter Strich auf der Timeline repräsentiert die aktuelle Position im Datensatz.

2. Session

Das Sessionfenster zeigt den aktuellen Datenpunkt an der Position an, wo der Teilnehmer auf den Bildschirm geschaut hat.

3. Aggregated Window

Zeigt alle Datenpunkte an, die sich im sogenannten Data-Window auf der Timeline befinden. Die Größe des Data-Window kann angepasst werden.

4. Distance Window

Gibt alle Distanzen der aufeinanderfolgenden Datenpunkte als Boxplot an, die sich im Data-Window Bereich befinden.

5. Velocity Window

Gibt alle Geschwindigkeiten der aufeinanderfolgenden Datenpunkte als Säulendiagramm an, die sich im Data-Window Bereich befinden.

Die aktuelle Position im Datensatz kann über die Tasten 'a' (zurück) und 'd' (weiter) verschoben werden.¹⁴ Und mithilfe der Tasten 'z' (Fixation), 'u' (Sakkade) und 'i' (Undefiniert) ist es möglich, die Label zu ändern. Mit diesem Programm hat der Autor die erste Session des Datensatzes nachbearbeitet. Das hat ungefähr eine halbe Stunde gedauert und bei einigen Datenpunkten ist eine Entscheidung schwer gefallen. Im Ergebnis haben sich die Zahlen deutlich verschoben:

	Sakkaden	Fixationen
Heuristik	400 (48 %)	441 (52 %)
Manuell	271 (32 %)	570 (68 %)

Die Anzahl der Fixationen ist gestiegen. Es wurde oben bereits diskutiert, dass die Heuristik an den Anfängen und Enden von Sakkaden zu unscharf ist. Dieses Problem wurde hier durch manuelles Nachbearbeiten beseitigt. Der Datensatz wird im folgenden Kapitel verwendet, um elementare Algorithmen zu testen.

3.4 Klassifizierung mit Algorithmen

Bisher hat die Arbeit gezeigt, dass sich in den WebGazer-Daten Fixations- und Sakkadengruppen visualisieren lassen. Dieses Kapitel testet nun einfache Algorithmen zur Klassifizierung, um zu prüfen, ob diese Gruppen auch mathematisch zu finden sind. Zemblys et. al. geben in ihrem Paper eine kurze Zusammenfassung der traditionellen Ansätze:

For a long time, two broad classes were used: First, the *velocity-based algorithms* that detect saccades and assume the rest to be fixations. [...] The *dispersion-based algorithms* instead detect fixations and assume the rest to be saccades (Zemblys u. a. 2018, S. 1)

Diesen beiden Klassen folgend hat der Autor je einen Algorithmus entwickelt, der mithilfe der Geschwindigkeit Sakkaden und mithilfe der Distanz Fixationen findet. Ein dritter Ansatz ist die Verwendung gerichteter Vektoren.

¹⁴Für eine ausführliche Beschreibung der Applikation sei auf die Dokumentation im Code verwiesen.

3.4.1 Algorithmus: Distanz

Bei diesem Ansatz werden Datenpunkte ausschließlich als *Fixationen* klassifiziert. Es werden eine bestimmte Anzahl (*windowsize*) (zeitlich) aufeinanderfolgende Punkte ausgewählt und die Distanzen aller Punkte zu einander berechnet (vgl. Algorithmus *calcAllDistances*). Liegt das Maximum aller Distanzen unter einem Schwellenwert (*threshold*), bedeutet das, dass alle Punkte dicht beisammen liegen. Dementsprechend werden alle Punkte dieser Gruppe als *Fixation* klassifiziert. Der erste Datenpunkt der Gruppe wird anschließend entfernt und der nächste, noch nicht in der Gruppe enthaltene, Datenpunkt hinzugefügt. Bei diesem Ansatz müssen zwei Parameter gesetzt werden: die Gruppengröße und der Grenzwert.

Algorithm 1 DistanceRollingWindow

Require: data : List GazeElement, threshold : Float, windowsize : Int

Ensure: List LabeledGazeElement

```

1:  $\forall e : e \in \text{data} \rightarrow \text{label } e \text{ as 'undefined'}$ 
2: for i := 1 TO length(data) - windowsize do
3:   currentPoints = {data[i]}  $\cup \dots \cup \{data[i + \text{windowsize} - 1]\}$ 
4:   distances = calcAllDistances( currentPoints )
5:   if max(distances) < threshold then
6:      $\forall e : e \in \text{currentPoints} \rightarrow \text{label } e \text{ as 'fixation'}$ 
7:   end if
8: end for
9: return data

```

Der Algorithmus 1 erzeugt zu Beginn für alle Daten ein Label, welches als *undefined* bezeichnet wird (Zeile 1). Eine For-Schleife (Zeile 2) läuft über alle Daten. Die Punkte der Gruppe bestehen aus den Daten von i bis $i + \text{windowsize}$ (Zeile 3) und von ihnen werden in Zeile 4 alle Distanzen berechnet. Der dazu verwendete Algorithmus 2 besteht in dem naiven Verfahren aus einer doppelten For-Schleife (Zeile 2,3), in der alle euklidischen Distanzen berechnet und als Liste zurückgegeben werden (Zeile 8).¹⁵ Befindet sich jetzt die maximale Distanz (Alg. 1, Zeile 5 - 7) unter einem gegebenen Schwellenwert (*threshold*), werden alle Punkte dieser Gruppe als *Fixation* gelabelt (Zeile 6).

Da die Distanzen aller Punkte zueinander berechnet werden und nicht nur die Distanzen zeitlich aufeinanderfolgender Punkte, ist die maximale Distanz abhängig von der Gruppengröße. Je mehr Punkte betrachtet werden, desto wahrscheinlicher liegen zwei (z. B. zeitlich weit entfernte) Punkte weiter auseinander. Allerdings gilt umgekehrt, dass je größer ein Gruppe ist, die nur kleine Distanzen enthält, die Punkte dieser Gruppe wahrscheinlich Fixationen sind. Wenn die Gruppengröße zu klein ist, könnten es auch nur einige verstreute Datenpunkte innerhalb einer Sakkade sein (vgl. etwa Abb. 8, (5)). Daran sieht man, dass die Wahl der Parameter eine wichtige Rolle spielt und Erfahrungen im Umgang mit den Daten braucht.

Um für den Algorithmus eine geeignete Gruppengröße und einen geeigneten Grenzwert zu finden, wurde mit einer *Brute-Force*-Methode alle Gruppengrößen von 3 bis 11 und jeweils alle Grenzwerte von 40 bis 200 ausprobiert. Das Ergebnis ist in Abbildung 11 zu sehen. In der

¹⁵Das Verfahren wird als naiv bezeichnet, weil die Distanzen von Punkten, die im nächsten Schleifenschritt in der Gruppe verbleiben, jedesmal neu berechnet werden. Eine effizientere Lösung ist denkbar.

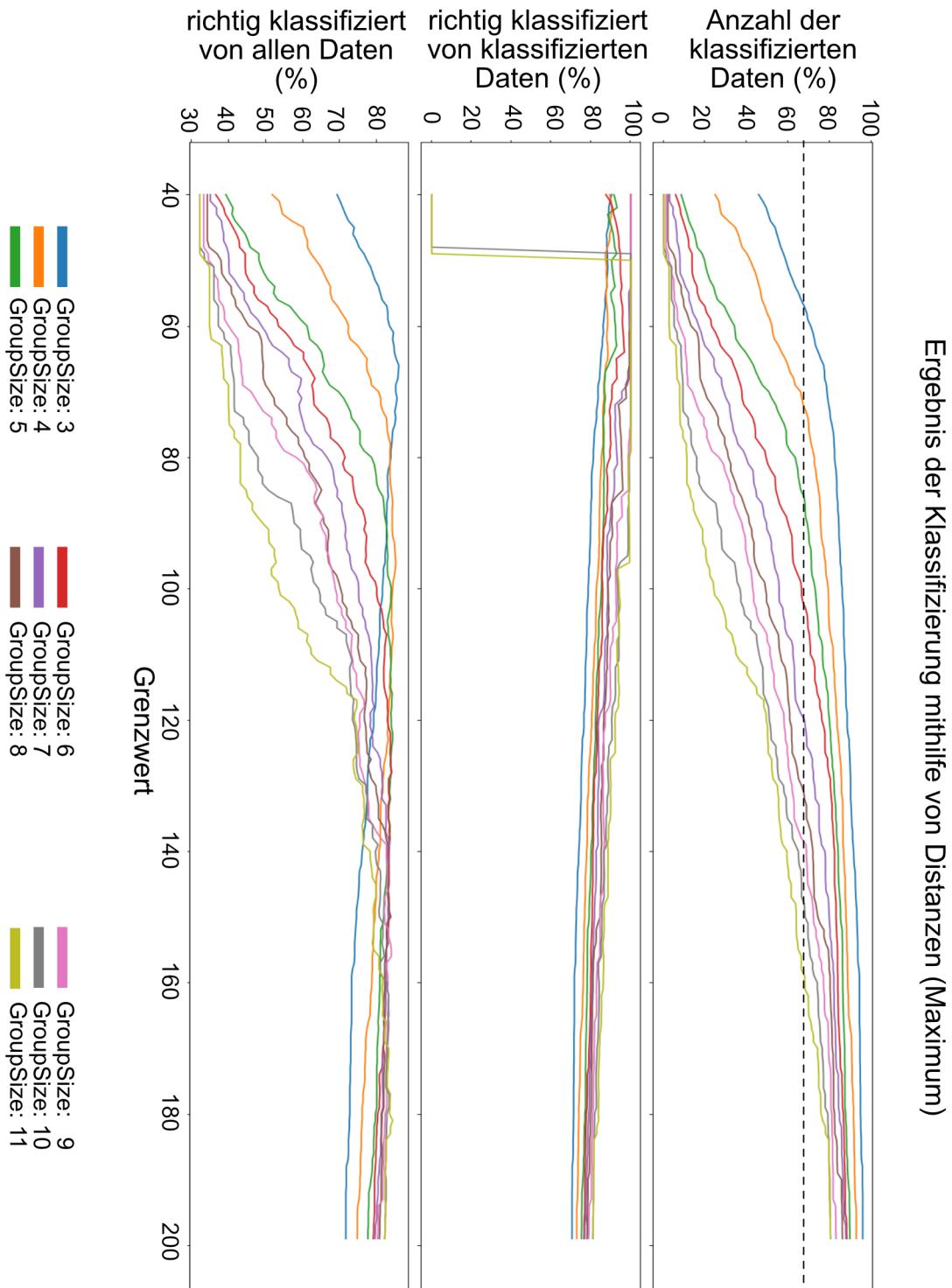


Abbildung 11: Ergebnis der Klassifizierung für verschiedene Grenzwerte und Gruppengrößen. Es wurde das Maximum jeder Gruppe mit dem Grenzwert verglichen. Die obere Grafik zeigt den Anteil klassifizierter Daten (%), die mittlere Grafik die von diesem Anteil richtig klassifizierten Daten (%) und die untere Grafik den Anteil der insgesamt 20 richtig klassifizierten Daten (%) an.

Algorithm 2 calcAllDistances

Require: el : List GazeElement**Ensure:** distances : List Float

```

1: result = []
2: for i := 1 TO length(el) - 1 do
3:   for j := i + 1 TO length(el) do
4:     euclideanDistance =  $\sqrt{(el[i].x - el[j].x)^2 + el[i].y - el[j].y)^2}$ 
5:     result.push(euclideanDistance)
6:   end for
7: end for
8: return result

```

oberen Darstellung wird der Anteil der klassifizierten Daten in Prozent angegeben, wobei die gestrichelte Linie den Anteil angibt, der mit Hand als Fixation klassifiziert wurde. Je größer der Grenzwert wird, desto mehr Daten werden klassifiziert. Bei einem sehr kleinen Grenzwert hingegen erfüllen nur wenige Gruppen das Kriterium, damit die Datenpunkte als Fixation gelabelt werden. Mit zunehmender Gruppengröße wird es unwahrscheinlicher, dass die maximale Distanz aller Punkte zueinander unter dem Grenzwert liegt. Bei einer Gruppengröße von 3 und einem sehr geringen Grenzwert von 40 Pixel werden knapp über 40 Prozent der Daten klassifiziert. Für die Gruppengrößen 10 und 11 wird gar kein Datenpunkt klassifiziert. Das lässt sich gut in der mittleren Grafik erkennen. Sie stellt dar, wie viel Prozent der klassifizierten Daten richtig (also auch im hand-gelabelten Datensatz als Fixation) klassifiziert wurden. Bei den letzten beiden Gruppengrößen werden überhaupt erst Daten ab einem Grenzwert von ungefähr 50 Pixel klassifiziert. Diese dann dafür zu 100 Prozent richtig. Generell gilt, je größer die Gruppengröße ist, um so weniger Daten werden klassifiziert, aber die die klassifiziert wurden, sind sehr wahrscheinlich korrekt als Fixationen erkannt.

Die untere Darstellung gibt an, wie viel Prozent der Daten richtig klassifiziert wurden, wenn nicht gelabelte Daten (bzw. als *undefined* gelabelt) zu Sakkaden erklärt werden. (Diese Heuristik ist der Annahme geschuldet, dass es in den Daten nur Fixationen und Sakkaden gibt.) Dabei schneiden kleine Gruppen nur mit einem kleinen Grenzwert gut ab. Auffällig ist vor allem, dass bei Gruppengröße 3 und 4 die besten Werte erst erreicht werden, nachdem bereits mehr Daten als Fixation klassifiziert wurden, als überhaupt im Datensatz vorhanden sind. Die Gruppengröße 5 scheint hier eine ganz gut Balance zu haben.

Im Ergebnis kann man sagen, dass sich aus der Grafik nicht eindeutig die beste Kombination aus den beiden Werten schlussfolgern lässt. Das bestätigt die in der Einleitung zitierte Einschätzung von Zemblys, dass es ein Vorteil von einem *Random Forest*-Klassifizierer ist, keine Parameter einstellen zu müssen.

Verwendet man anstatt des Maximums den Durchschnitt, muss der Grenzwert deutlich geringer angesetzt werden. Es geht dann nicht mehr darum, dass eine Menge von Punkten keinen Ausreißer hat, sondern die Punkte im Durchschnitt eng beieinander liegen. Ein Ausreißer kann hier durch besonders verdichtete Punkte ausgeglichen werden.

Da sich aus den oberen Grafiken keine eindeutige Kombination von Grenzwert und Schwellenwert ablesen lässt, sollen hier zwei Varianten ausprobiert und die Ergebnisse visualisiert werden. Es wird eine kleine Gruppengröße von drei mit einem kleinen Schwellenwert von 55 und eine große Gruppengröße 10 mit einem großen Schwellenwert von 145 verwendet. Diese

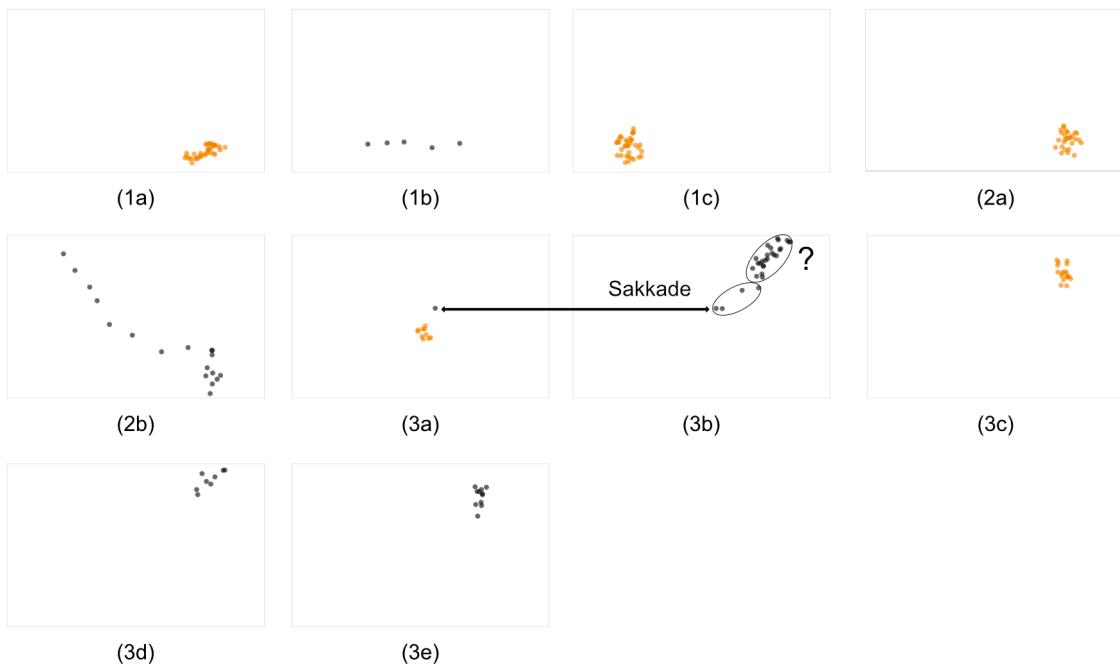


Abbildung 12: Ergebnis des Distanzalgorithmus mit einer Fenstergröße von 10 und einem Schwellenwert von 145 Pixel.

Auswahl deckt die beiden unterschiedlichen Spektren ab. Verfolgt wird jeweils der Ansatz mit dem Maximum.

Abbildung 12 zeigt das Ergebnis, wenn der Algorithmus mit einer Gruppengröße von 10 und einem Schwellenwert von 145 angewendet wird. Allgemein lässt sich feststellen, dass der Algorithmus funktioniert. Er kann große Gruppen von Punkten (Fixationen) von einzelnen Punkten (Sakkaden) unterscheiden (1a - 1c).

Der exakte Anfang der Sakkade kann dabei verloren gehen (2 a - 2b). An dieser Stelle liegt das vor allem daran, dass einige Datenpunkte aus dem unteren Bildschirmrand verschwinden und wieder zurückkehren. Dennoch wird hier eigentlich der untere rechte Bildschirmrand fixiert, bevor es zum nächsten Zielpunkt in die obere linke Ecke geht.

Obwohl der Schwellenwert mit 145 recht groß gewählt ist, kann es trotzdem passieren, dass größere Gruppen von Punkten, die eindeutig eine Fixation bilden nicht als solche erkannt werden (3b). Aber es stimmt die generelle Richtung: Aus einer Fixation (3a) startend, gibt es eine kurze Sakkade in die obere rechte Bildschirmecke, die letztendlich in der nächsten Fixation (3c) mündet. Das diese große Gruppe nicht als Fixation klassifiziert wird, liegt einerseits daran, dass wir am oberen rechten Bildschirmrand wieder einige Datenverluste haben (3d). Dadurch enthalten die Gruppen teilweise Bildpunkte, die nicht mehr auf dem Bildschirm sichtbar und extrem weit weg sind. Andererseits liegt es auch einfach an dem gewählten Schwellenwert. Die Punktwolke in (3e) enthält einen maximalen Distanzwert von 150,71 und ist damit um knapp 6 Pixel über dem Schwellenwert.

3.4.2 Algorithmus: Geschwindigkeit

Der hier gewählte Ansatz ähnelt dem Vorhergehenden: Die Punkte werden wieder gruppiert, aber diesmal werden nicht die Distanzen sondern die Geschwindigkeiten für die Klassifizierung verwendet. Wurden bei den Distanzen die Datenpunkte, die beieinander lagen, als Fixationen bezeichnet, werden im folgenden Algorithmus die Daten als Sakkaden bestimmt, wenn die Geschwindigkeiten zwischen Punkten hinreichend groß ist.

Der Algorithmus 3¹⁶ setzt zunächst alle Label auf *undefined* [1]. In der folgenden For-Schleife [2-8] werden eine bestimmte (*windowsize*) Anzahl von aufeinanderfolgenden Datenpunkten zu einer Gruppe (*currentPoints* [3]) zusammengefasst. Während der Distanzalgorithmus alle möglichen Distanzen innerhalb der Gruppe berechnet, werden bei diesem Ansatz nur die Geschwindigkeiten am Rand der Gruppe genommen (vgl. Algorithmus 4). Die Annahme ist, dass die Punkte innerhalb der beiden Randpunkte ebenfalls zur Sakkade gehören, auch wenn ihre Geschwindigkeiten manchmal etwas geringer ausfallen.

Algorithm 3 VelocityRollingWindow

Require: data : List GazeElement, treshold : Float, windowsize : Int

Ensure: List LabeledGazeElement

```

1:  $\forall e : e \in \text{data} \rightarrow \text{label } e \text{ as } \text{'undefined'}$ 
2: for i := 1 TO length(data) - windowsize do
3:   currentPoints = {data[i]}  $\cup \dots \cup \{data[i + \text{windowsize}]\}$ 
4:   borderVelocities = calcBorderVelocities( currentPoints )
5:   if min(velocities) > treshold then
6:      $\forall e : e \in \text{currentPoints} \rightarrow \text{label } e \text{ as } \text{'saccade'}$ 
7:   end if
8: end for
9: return data

```

Für diesen Algorithmus müssen zwei Parameter gesetzt werden. Die Brute-Force-Methode für optimale Parameter ist wie beim Distanzalgorithmus unergiebig. Deshalb wurde mit verschiedenen Werten experimentiert. Das repräsentativ beste Ergebnis ist in Abbildung 13 zu sehen. Die Fenstergröße ist 4 und der Schwellenwert beträgt 1,5 (Pixel pro ms). Dieses Beispiel zeigt, dass Sakkaden mit dem Algorithmus gefunden werden (1). Die nicht klassifizierten Gruppen bilden deutliche Fixationen (3). Dass die Geschwindigkeiten nur am Rand berechnet werden und nicht von allen aufeinanderfolgenden Datenpunkten, hat den Vorteil, dass innerhalb einer Sakkaden auch Punkte dichter beisammen sein können (2). Aber nur weil Punkte sich häufen (4), muss es sich nicht um eine Fixation handeln. Bei der Darstellung (4) handelt es sich tatsächlich um Augenbewegungen, die sich hin und her bewegen und wenig konstant sind. Die Parameter können für bestimmte Konstellationen ungünstig sein (5). So reicht der Schwellenwert knapp nicht, um die vier Datenpunkte zwischen zwei vermeintlichen Fixationen zu erkennen.

¹⁶Im Algorithmus sind die Zeilen rot gefärbt, die sich vom Distanzalgorithmus unterscheiden

Algorithm 4 calcBorderVelocities

Require: el : List GazeElement
Ensure: velocities : List Float

- 1: result = []
- 2:
- 3: firstEl = el[1]
- 4: secondEl = el[2]
- 5: euclideanDistance = $\sqrt{(firstEl.x - secondEl.x)^2 + (firstEl.y - secondEl.y)^2}$
- 6: time = secondEl.timestamp - firstEl.timestamp
- 7: result.push(euclideanDistance / time)
- 8:
- 9: beforeLastEl = el[length(el) - 1]
- 10: lastEl = el[length(el)]
- 11: euclideanDistance = $\sqrt{(beforeLastEl.x - lastEl.x)^2 + (beforeLastEl.y - lastEl.y)^2}$
- 12: time = lastEl.timestamp - beforeLastEl.timestamp
- 13: result.push(euclideanDistance / time)
- 14:
- 15: **return** result

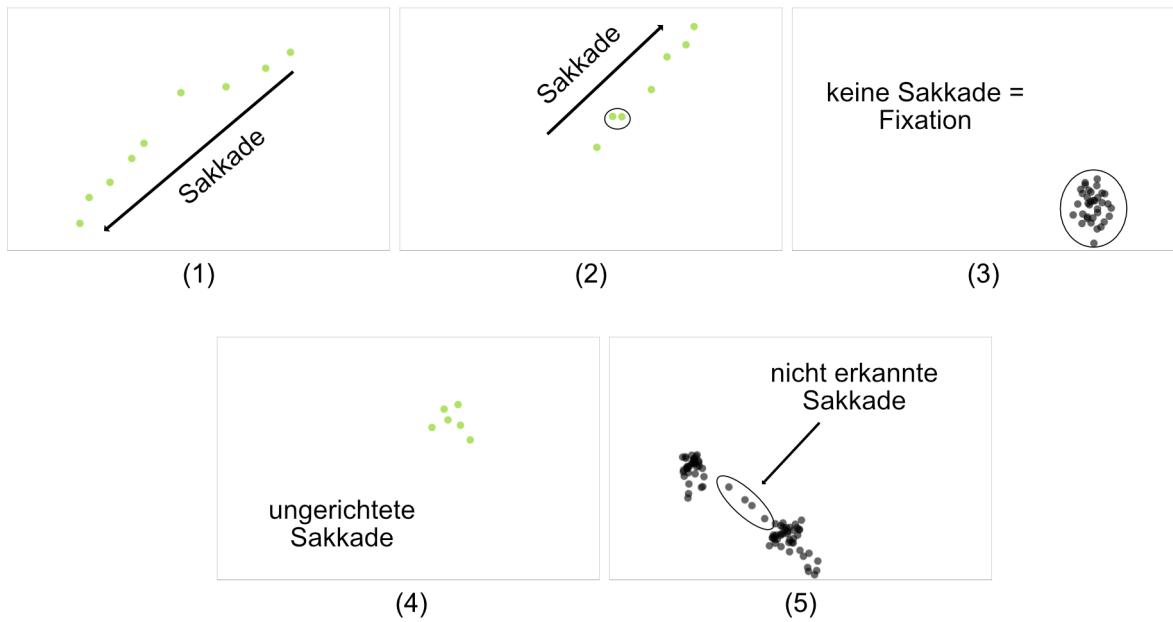


Abbildung 13: Ergebnis des Geschwindigkeitsalgorithmus mit einer Fenstergröße von 4 und einem Schwellenwert von 1,5 (Pixel pro ms).

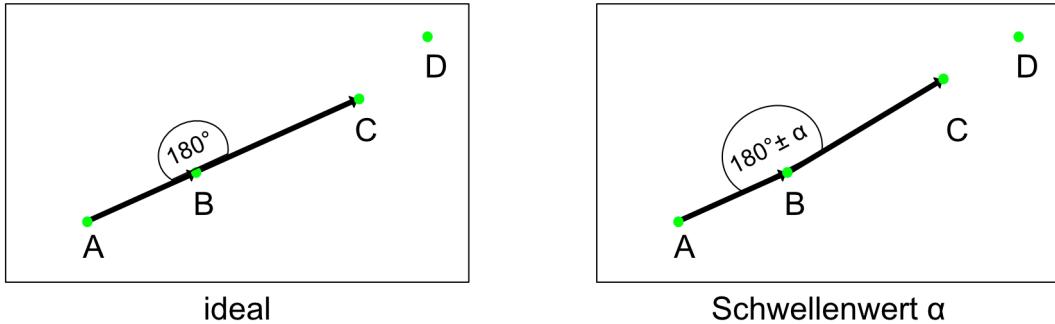


Abbildung 14: Grundidee des Vektor-Algorithmus.

3.4.3 Algorithmus: Vektor

Neben der Geschwindigkeit und der Distanz zwischen Datenpunkten ist die Richtung ein Anhaltspunkt für Sakkaden. Bewegen sich die Datenpunkte auf einer Linie, so ist zu vermuten, dass der Nutzer von einer Fixation zur nächsten geht. Haben die Punkte hingegen keine feste Richtung oder springen sie gar um ein Zentrum hin und her, dann wird etwas fixiert. Der Ansatz versucht also, über die Richtung der Vektoren zu bestimmen, ob die Datenpunkte auf einer ungefähren Linie liegen. Er benötigt ebenfalls einen Parameter, der festlegt, wie stark eine Linie gekrümmmt sein darf.

Abbildung 14 zeigt das Vorgehen des Algorithmus. Von jeweils drei Datenpunkten A, B, C werden die beiden Vektoren \vec{AB} und \vec{BC} gebildet. Beträgt der Winkel in B genau 180° , dann liegen die drei Punkte auf einer exakten Linie. Da dieser Fall kaum eintreten wird, kann über einen Parameter α justiert werden, wie weit die Linie gekrümmmt sein darf. Erkennt der Algorithmus die drei Punkte als auf einer Linie liegend an, dann werden die ersten beiden Punkte (A und B) als Sakkade klassifiziert und die nächsten drei Punkte B, C und D betrachtet. Erst im nächsten Schritt wird also der Punkt C als Sakkade klassifiziert, wenn sich herausstellt, dass die Bewegung in einer Richtung fortgesetzt wird. Bricht hingegen die Bewegung ab, ist die Annahme, dass Punkt C der erste Punkt einer Fixation ist.

Der Algorithmus 5 läuft mit einer For-Schleife über alle Daten, wobei die letzten beiden Datenpunkte ausgespart werden [2]. Der Grund dafür ist, dass vom aktuellen Datenpunkt i aus [2], die beiden folgenden Punkte $i + 1$ und $i + 2$ für die Berechnung hinzugezogen werden [3, 4]. Von diesen drei (x, y) Koordinaten bildet der Algorithmus 6 zwei Vektoren ausgehend vom zweiten Punkt [2,3] und berechnet den Winkel zwischen diesen Vektoren [5-8]. Der Algorithmus 5 benutzt diesen Winkel, um den Schwellenwert mit der Differenz bis 180° zu vergleichen [6]. Wenn die Differenz klein genug ist, bezeichnet der Algorithmus die Datenpunkte i und $i + 1$ als Sakkaden. Das dritte Datum $i + 2$ wird gegebenenfalls erst im nächsten For-Schritt klassifiziert. Abbildung 14 verdeutlicht dieses Vorgehen visuell. Zuerst betrachtet der Algorithmus die Punkte A, B und C. Ist α klein genug, bezeichnet er nur A und B als Sakkaden, denn C könnte bereits der Beginn einer Fixation sein, wenn der folgende Punkt D etwa in eine völlig andere Richtung geht. Im folgenden Schleifenschritt untersucht

Algorithm 5 Labeln von Sakkaden mittels der Richtung von Vektoren

Require: data : List GazeElement, treshold : Float**Ensure:** List LabeledGazeElement

```

1: for i := 1 TO length(data) - 2 do
2:   p1 = data[i]
3:   p2 = data[i + 1]
4:   p3 = data[i + 2]
5:   angle = calcAngle(p1, p2, p3)
6:   if | angle - 180 | < treshold then
7:     data[i] labeled as 'saccade'
8:     data[i + 1] labeled as 'saccade'
9:   end if
10: end for
11: return data

```

Algorithm 6 Berechnung des Winkels zwischen zwei Vektoren

Require: p1 : (x, y), p2 : (x, y), p3 : (x, y)**Ensure:** Angle in ° : Float

```

1:
2: vector1 = (p1.x - p2.x, p1.y - p2.y)
3: vector2 = (p3.x - p2.x, p3.y - p3.y)
4:
5: scalarProduct =  $\overrightarrow{\text{vector1}} \cdot \overrightarrow{\text{vector2}}$ 
6: lengthVector1 = | vector1 |
7: lengthVector2 = | vector2 |
8: radians =  $\cos^{-1} \left( \frac{\text{scalarProduct}}{\text{lengthVector1} \cdot \text{lengthVector2}} \right)$ 
9:
10: return toDegree(radians)

```

der Algorithmus dann die Punkte B, C und D, wobei C zum ersten mal klassifiziert werden könnte.

In Abbildung 15 sind die Ergebnisse des Algorithmus mit einem Grenzwert von 10° also einem zulässigen Bereich von $170^\circ - 190^\circ$ dargestellt. (1) zeigt, dass der Algorithmus prinzipiell (längere) Sakkaden finden kann. Die Bewegung ist vom unteren rechten Bildschirmbereich zur oberen linken Ecke. Anschließend sind zwar einige Datenpunkte nicht vorhanden, weil sie außerhalb des Bildschirmbereichs liegen, aber folgende Punkte zeigen, dass sich eine (zumindest kleine) Fixation anschließt. Von diesen langen Sakkaden gibt es allerdings nur zwei im Datensatz. Eine längere Sakkade, bei der der Grenzwert nicht ausreichend ist, zeigt die zweite Darstellung (2). Hier findet die Bewegung von oben rechts nach unten links statt, an die sich eine Fixation anschließt. Während die beiden Punkte direkt vor dem Cluster nur knapp den Grenzwert ($|\alpha| \approx 11.39^\circ$) überschreiten, liegt in der Mitte der Sakkade ein Datenpunkt, der sowohl in der ersten ($|\alpha| \approx 15.92^\circ$) als auch in der zweiten ($|\alpha| \approx 43.97^\circ$) Dreiergruppe deutlich den Grenzwert überschreitet (2a).

Doch meist findet der Algorithmus nur kleine Gruppen von Punkten, die auf einer Linie liegen (3). Zwar mag es so aussehen, als würde in (3a) eine Sakkade weitergehen, doch (3b) zeigt, dass der nächste Datenpunkt abrupt in der entgegengesetzten Richtung liegt. Das liegt an der Eigenheit der Daten, die WebGazer liefert. Da selbst bei Fixationen die Datenpunkte nicht auf einer Koordinate verbleiben, sondern ständig in Bewegung sind, ergeben sich immer wieder kleinere Bewegungen in eine Richtung, die so wirken, als würde die Blickrichtung sich langsam verändern (4), dabei gehören die Punkte zu Daten, die zusammen betrachtet, eine Fixation ergeben (4a).

Es ergibt sich ein weiteres Problem, wenn Datenpunkte sehr dicht beisammen liegen (4). In diesem Fall haben bereits kleine Abweichungen großen Einfluss auf den Winkel. Gibt es also in einer Sakkade Punkte, die dicht beieinander sind, dann werden sie mit hoher Wahrscheinlichkeit den Grenzwert überschreiten.

Zwar braucht der Ansatz mit Vektoren nur einen Parameter, allerdings ist er von den drei vorgestellten Algorithmen der unzuverlässigste. Eine Verknüpfung der Ansätze könnte bessere Klassifizierungsergebnisse bringen.

3.5 Zwischenergebnis

Dieses Kapitel hat gezeigt, dass WebGazer-Daten erkennbare Fixationen und Sakkaden enthalten können. Die Abbildungen 7 und 9 visualisieren Daten einer ganzen Session. Die Methode zur Datenaufnahme leitet den Nutzer in seinen Handlungen an und sorgt so für eine klare Trennung zwischen Fixationen und Sakkaden. Die verwendete Heuristik klassifiziert die Datenpunkte im Groben richtig. Besonders die Fixationsgruppen sind als solche eindeutig. Bei den Sakkadengruppen hingegen finden sich Fixationspunkte vom Anfang und Ende der Sakkade. Dieser *onset* bzw. *offset* einer Fixation, also der exakte Beginn und das exakte Ende einer Fixation, können durch die Heuristik nicht erfasst werden. Eine manuelle Nachbearbeitung der Daten hat signifikanten Einfluss auf die Anzahl der Fixationen und Sakkaden.

Algorithmen, welche die Distanzen, Geschwindigkeiten oder Richtungen der Vektoren verwenden, finden die Fixations- und Sakkadengruppen in den Daten. Dabei bestätigt sich die in der Einleitung angedeutete Schwierigkeit, geeignete Parameter zu wählen. Ob diese Algorithmen sinnvolle Klassifikationen von Daten erzeugen, die nicht in der Datenerhebung so auf Fixationen und Sakkaden ausgerichtet sind, bleibt zu testen.

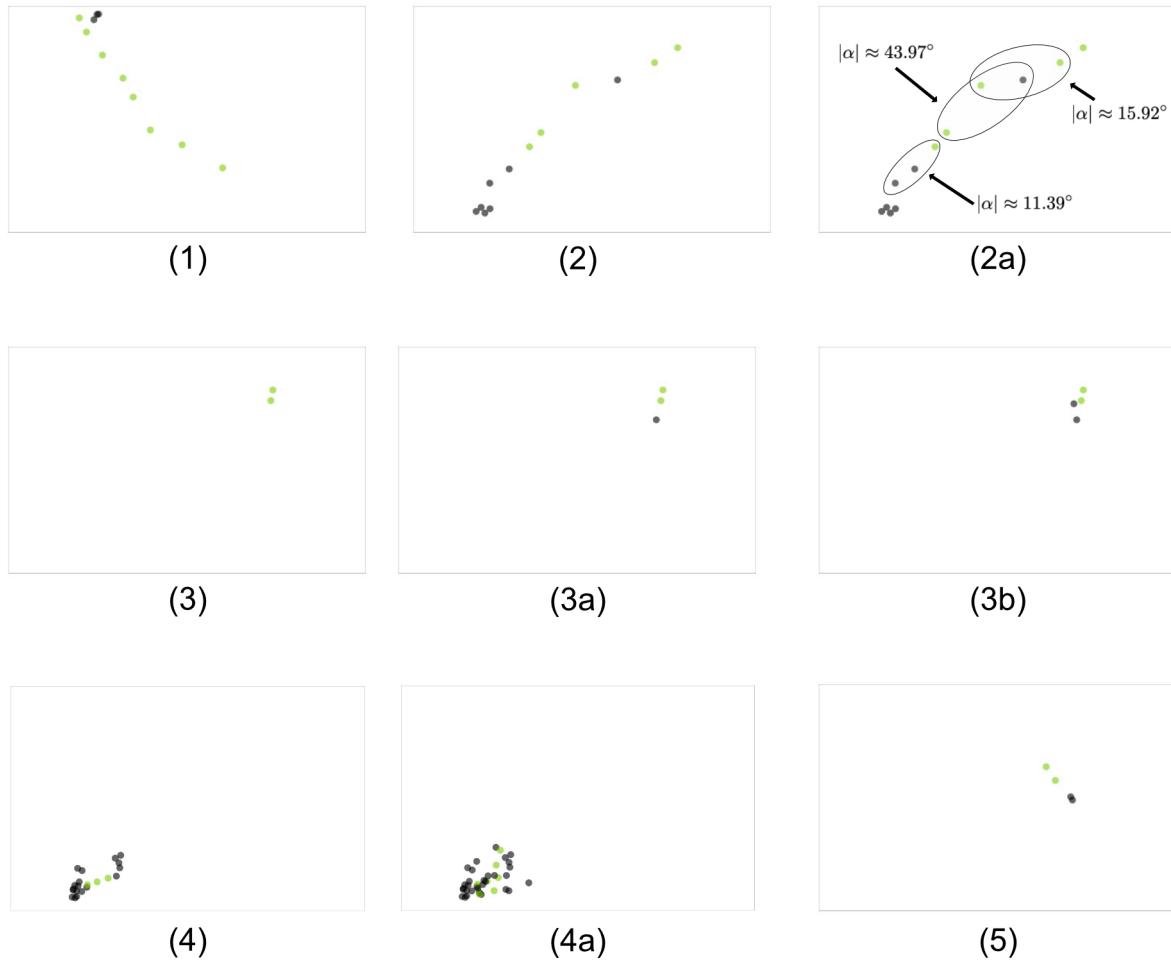


Abbildung 15: Ergebnis des Vektor-Algorithmus mit: $threshold = 10^\circ$. (1) zeigt die prinzipielle Funktionsweise mit kleinen Problemen in (2). Der Algorithmus produziert sehr viele kurze Sakkaden (3)(4)(5).

Es stellt sich die Frage, ob der in diesem Kapitel vorgestellt Datensatz für maschinelles Lernen verwendet werden kann. Die Datenaufnahme wäre einfach und würde in kurzer Zeit viele Daten erzeugen, die ein Modell trainieren könnten. Dagegen spricht, dass die Heuristik nicht genau genug ist. Für die manuelle Nachbearbeitung der Daten, die zwingend nötig wäre, steht kein Experte zur Verfügung. Zudem ist sie zeitaufwendig. Vielleicht ließe sich die Aufgabe mithilfe der WebGazer-Community lösen. Gemeinsam könnte man den Zeitaufwand teilen und Experten hinzuziehen. Doch es ist zweifelhaft, ob sich dieser Aufwand lohnt, denn die Daten entstehen unter künstlichen Bedingungen. Ein gewöhnlicher Anwendungsfall von WebGazer erzeugt solche Daten nicht, weil der Nutzer nicht auf bestimmte Handlungen festgelegt ist. Somit bliebe die Frage, welchen Mehrwert die Daten wirklich hätten.

Mit dieser Problematik, einen klassifizierten Datensatz von WebGazer unter realen Bedingungen zu erstellen, beschäftigt sich das folgende Kapitel.

4 Erstellen eines klassifizierten WebGazer-Datensatzes

Nachdem im 2. Kapitel gezeigt wurde, dass sich WebGazer-Daten sinnvoll in Fixationen und Sakkaden klassifizieren lassen, geht es in diesem Kapitel darum, einen WebGazer-Datensatz zu erstellen, der unter realen Bedingungen aufgenommen wird und klassifizierte Daten enthält. Zu den realen Bedingungen in diesem Kapitel gehört das Lesen eines Textes, das Ansehen einer Grafik und das Auswählen unter Optionen. Gelingt es, einen solchen klassifizierten Datensatz zu erstellen, bietet dieser die Möglichkeit, einen *Random Forest*-Klassifizierer zu trainieren.

4.1 Methode

Das Ziel der Datenerhebung ist es, klassifizierte WebGazer-Daten zu erhalten. Dazu soll in einem Experiment, Daten von WebGazer und von Eyetracking-Hardware gleichzeitig aufgenommen werden. Als Hardwaresystem steht das Tobii Pro X3-120¹⁷ mit einer externen Prozesseinheit¹⁸ zur Verfügung. Das zugehörige Softwaresystem klassifiziert die Datenpunkte als *Fixation*, *Saccade* oder *Undefined*. Beide Datensätze (Hardware und WebGazer) sollen anschließend zusammengeführt werden, um die Label der Hardwaredaten auf die von WebGazer zu übertragen.

4.1.1 Inhalt

Der zentrale Teil der Studie ist eine Grafik, die der Teilnehmer betrachten soll. Aber um mehr Daten aus verschiedenen Anwendungsfeldern zu sammeln, werden noch zwei Aufgaben hinzugefügt. Vor der Grafik erhält der Teilnehmer eine Aufgabenbeschreibung, die ihm das weitere Vorgehen erklärt und an die Grafik anschließend kommt eine Multiple-Choice-Frage zu der Grafik.

Im Rahmen dieser BA-Arbeit wird die Studie am LLZ¹⁹ durchgeführt. Dort gibt es einen Computer, bei dem die Hardware und Software des Tobii Pro Systems bereits installiert ist. Deshalb werden alle Studienteilnehmer am selben Bildschirm arbeiten. Trotzdem werden die

¹⁷<https://www.tobiipro.com/product-listing/tobii-pro-x3-120/>, abgerufen am 01.12.2020.

¹⁸<https://www.tobiipro.com/product-listing/external-processing-unit/>, abgerufen am 01.12.2020.

¹⁹<https://www.llz.uni-halle.de/>, abgerufen am 20.12.2020.

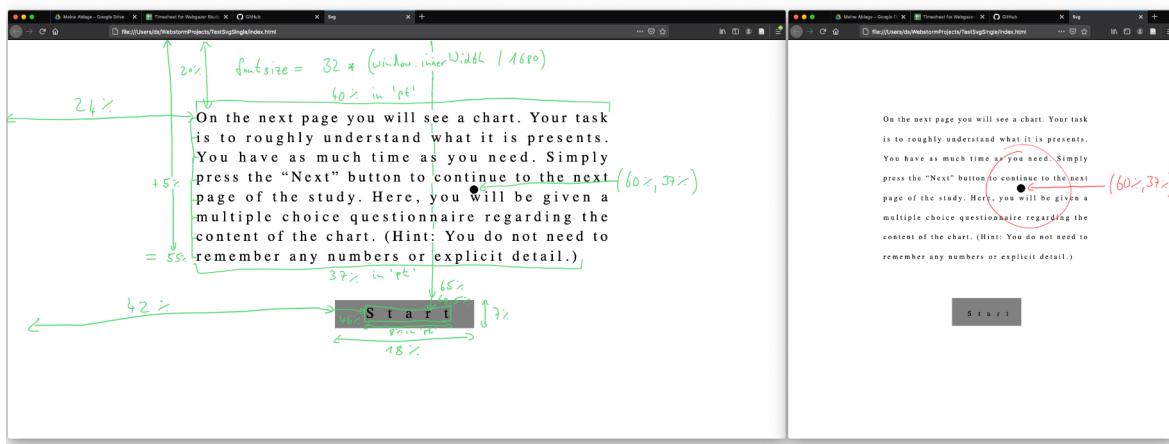


Abbildung 16: Stimulus 1 - Aufgabenbeschreibung. Die rechte Seite ist dieselbe Abbildung mit nur der halben Bildschirmgröße. Man beachte, dass der hypothetische Datenpunkt bei (60 %, 37 %) jeweils an der exakt selben Position im Text ist.

im folgenden beschriebenen Inhalte nicht für diese Bildschirmgröße und Bildschirmauflösung optimiert, weil geplant ist, das Experiment nach Abgabe der BA-Arbeit im Internet allgemein zugänglich zu machen. Die so erzeugten Daten lassen sich zwar nicht mit Hardwaredaten synchronisieren oder vergleichen, können aber wiederum mit den WebGazer-Daten aus dieser BA-Arbeit verglichen werden. Daran knüpfen sich weitere Analysemöglichkeiten an, auf die hier nicht weiter eingegangen werden kann. Für die Inhalte bedeutet das, dass sie so konzipiert werden müssen, dass sie auf unterschiedlichen Fenstergrößen und unterschiedlichen Auflösungen funktionieren und die gesammelten Daten vergleichbar sind. Deshalb wird mit relativen Werten gearbeitet.

Stimulus 1 - Aufgabenbeschreibung

Vor der eigentlichen Grafik wird ein Text mit der Aufgabenstellung angezeigt (vgl. Abb. 16). Der Text ist auf Englisch und lautet:

On the next page you will see a chart. Your task is to roughly understand what it is presents. You have as much time as you need. Simply press the "Next" button to continue to the next page of the study. Here, you will be given a multiple choice questionnaire regarding the content of the chart. (Hint: You do not need to remember any numbers or explicit detail.)

Bereits hier werden die ersten Daten gesammelt, wobei das vorher den Teilnehmern nicht explizit gesagt wird. Der Text klärt darüber auf, dass im nächsten Schritt eine Grafik angezeigt wird, die der Teilnehmer verstehen soll. Es gibt keine Zeitbeschränkung. Mithilfe eines Buttons kann er zum nächsten Schritt gelangen, wo ihm eine (einfache) Multiple-Choice-Frage gestellt wird. Diese Frage soll die Teilnehmer motivieren, den Inhalt der Grafik wirklich zu verstehen. Andererseits gibt der Hinweis an, dass man sich nicht in Details verlieren soll.

Die Abbildung 16 zeigt zweimal den gleichen Stimulus, wobei die rechte Darstellung die Hälfte der Fensterbreite von der linken Darstellung hat. Der Text ist in beiden Fällen exakt 24 Prozent vom rechten und 20 Prozent vom oberen Bildschirmrand entfernt. Die Schriftgröße wird relativ zur Fensterbreite definiert als

$$\text{fontsize} = 32 \cdot \frac{\text{window.innerWidth}}{1680}.$$

Auch die Textbreite ist genau vorgegeben. Sie beträgt 40 Prozent. Nur die letzte Zeile wurde auf 37 Prozent gesetzt, weil ansonsten der Text sehr verzerrt ausgesehen hätte. Diese Darstellung macht den ganzen Text für verschiedene Monitorgröße variabel, ohne das sich durch automatische Zeilenumbrüche Worte im Text verschieben. Man beachte bspw. den hypothetischen Datenpunkt an der Koordinate (60 %, 37 %), der jeweils zwischen den Worten *continue* und *will* steht. Dieser Punkt verschiebt sich nicht, wenn verschiedene Fenstergrößen zum Einsatz kommen. Dadurch lässt sich später zeigen, dass ein Teilnehmer unabhängig von seinem verwendeten System auf einen bestimmten Punkt geschaut hat, wenn die Daten anhand der Bildschirmbreite und -höhe normalisiert sind. Der Nachteil ist, dass der Text vom Layout künstlich aussieht. Ein Studienteilnehmer hat sich darüber beschwert und vermutet, dass die ungewöhnliche Darstellung, die sich schwer lesen lassen würde, Teil der Studie sei.

Wenn man in Abbildung 16 beide Darstellungen vergleicht, fällt auf, dass es unterschiedlich lange Augenbewegungen braucht, um eine Zeile zu lesen. Das könnte die Daten schwieriger vergleichbar machen. Meine Vermutung ist allerdings, dass im Alltag die Bildschirmgröße mit der Entfernung des Auges vom Bildschirm korreliert. Verwendet also jemand einen kleinen Bildschirm, sitzt er auch näher davor. Verwendet hingegen jemand einen großen Bildschirm und sitzt trotzdem nicht weiter weg, wird er den Browser vermutlich nicht im Vollbildschirmmodus nutzen. Das Verhältnis von Bildschirmgröße und Distanz des Nutzers sollte die Länge von Augenbewegungen wieder ausgleichen.²⁰

Neben dem Text gibt es auf der Seite auch noch einen Button mit relativen Maßeinheiten. Diese Schaltfläche muss von allen Teilnehmer angeklickt werden (im Gegensatz dazu muss nicht jeder Teilnehmer den kompletten Text lesen). Deshalb wäre diese Fläche (unter semantischem Aspekt) eine AOI (area of interest) für die Analyse.

Stimulus 2 - Grafik

Die Grafik²¹ (Abb. 17) ist ein statisches Bild, das in der Mitte des Bildschirms platziert ist. Inhaltlich zeigt sie den Anstieg der Milliardäre sowie den Anstieg derer Vermögen in den Jahren 1987 bis 2013. Die Grafik hat einige markante Punkte, die vom Betrachter fixiert werden könnten.

Sie hat mehrere Textelemente:

- Eine fett gedruckte Überschrift die, bei der groben Orientierung der Grafik hilft.

²⁰Das sind natürlich hypothetische Annahmen, die bei einer Analyse von öffentlich gesammelten Daten getestet werden müsste. Abweichungen im Verhalten ließen sich etwa im Kalibrierungsschritt überprüfen, wo alle Probanden das selbe Muster von Daten ausweisen sollten.

²¹Die Grafik stammt aus dem Buch *Das Kapital im 21. Jahrhundert* (Piketty 2014) und ist online verfügbar unter: <http://piketty.pse.ens.fr/files/capital21c/en/pdf/F12.1.pdf>, abgerufen am 20.12.2020.

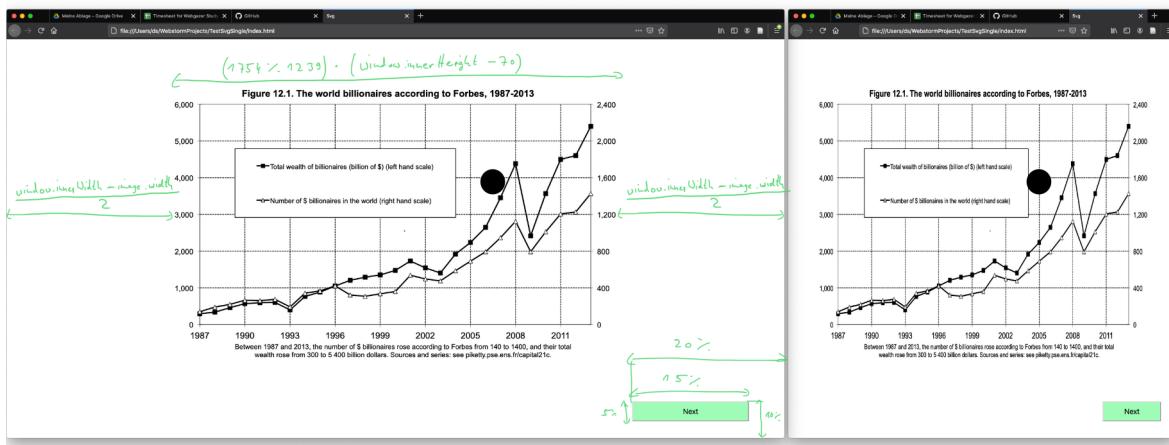


Abbildung 17: Stimulus 2 - Grafik. Der Plot ist ein Bild in JPEG-Format und ist deshalb nicht so flexibel anpassbar. Auf der rechten Seite ist die Abbildung bei halbierter Fenstergröße zu sehen.

- Eine zentral in die Grafik eingebaute Legende, die notwendig ist, wenn man die beiden Linien unterscheiden möchte.
- Eine Bildunterschrift, die aufgrund der kleinen Schrift unscheinbar wirkt, aber die Grafik im Wesentlichen verbal beschreibt.
- Drei Achsenbeschriftungen mit Zahlen.

Die beiden Linien haben eine klar ansteigende Richtung. Trotzdem sind sie nicht ohne weiteres zu verstehen. Einerseits muss anhand der Legende zunächst festgestellt werden, was jede Linie bedeutet, andererseits haben beide Linien auch unterschiedliche y-Skalen. Die rechte y-Skala gibt die Anzahl der Milliardäre auf der Welt an und gehört zu der Linie mit den Dreiecken. Die linke y-Skala hingegen skaliert das gesamte Vermögen (in Milliarden) der Milliardäre. Es bedarf also schon einiger Blicke, um die Aussage des Graphen zu verstehen.

Im unteren rechten Bildschirmrand wird ein Button angezeigt, der den Teilnehmer auf die nächste Seite bringt.

Das Bild ist im JPEG-Format mit einer Pixelgröße von 1232 x 802. Auf einem durchschnittlichen Bildschirm sollte die Grafik ohne Anpassungen in Originalgröße angezeigt werden können. Das Bild wird dabei immer in der Mitte platziert, wobei die (x, y) Koordinate des oberen linken Punktes gespeichert wird. Problematisch wird es dann, wenn ein sehr kleiner Bildschirm verwendet wird (mit evtl. geringer Auflösung). In diesem Fall wird das Bild gestaucht (vgl. Abbildung 17, rechts). Im Gegensatz also zum ersten und dritten Stimulus sind damit zwei prozentual gleiche Datenpunkt auf unterschiedlichen großen Bildschirmen, nicht mehr semantisch miteinander verbunden (vgl. den hypothetischen Blickpunkt in Abbildung 17). Für diese BA-Arbeit spielt das keine Rolle, weil die Bildschirmgröße für alle Teilnehmer gleich groß ist und sich die Grafik immer am selben Ort befindet.

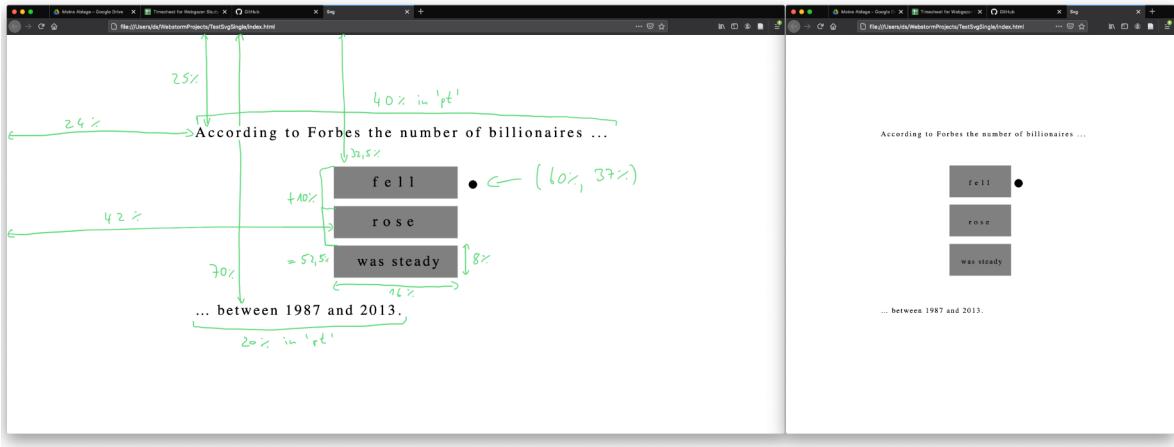


Abbildung 18: Stimulus 3 - Multiple-Choice-Frage. Auf der rechten Seite ist der selbe Inhalt bei halbierter Fenstergröße dargestellt. Man beachte, dass sich der hypothetische Datenpunkte (60 %, 37 %) bei beiden Darstellungen an der selben Stelle befindet.

Stimulus 3 - Multiple-Choice-Frage

Im letzten Schritt wird der Teilnehmer dazu aufgefordert, eine Frage zu der Grafik zu beantworten (vgl. Abb. 18). Sie lautet:

According to Forbes the number of billionaires...
 fell
 rose
 was steady
... between 1987 and 2013.

Die Frage ist mit Absicht leicht gehalten, dass sie auch nach einem kurzen Blick auf die Grafik beantwortet werden kann. Die Antwort ergibt sich intuitiv, weil beide Kurven in der Grafik ansteigen. Zusätzlich ist die Frage direkt der Bildunterschrift der Grafik entnommen. Keiner der Teilnehmer hatte eine Schwierigkeit, die Frage zu beantworten. Vielmehr waren einige enttäuscht, dass das ganze Experiment schon nach so kurzer Zeit zu Ende war. Vermutlich hätten sich einige noch ein paar mehr herausfordernde Fragen gewünscht.

Die Multiple-Choice-Frage besteht aus drei Buttons (die Auswahlmöglichkeiten) und einen umrahmenden Text. Alles ist zentral in der Mitte des Bildschirms ausgerichtet. Wie bei dem ersten Stimulus ist auch hier alles in relativen Größen formiert, so dass der hypothetische Blickpunkt in Abbildung 18 unabhängig von der Fenstergröße an der selben Stelle ist.

Es wird gespeichert, ob die Frage richtig beantwortet wurde. In diesem Fall würde man vermehrt Datenpunkte auf dem zweiten Button erwarten. Zusätzlich dient diese Frage auch als Filter. Wer die Frage falsch beantwortet hat, war bei der Grafik auf der vorherigen Seite nicht aufmerksam und die Daten haben deshalb möglicherweise weniger Aussagekraft.

4.1.2 Aufbau

WebGazer wurde mit der Idee entwickelt, Eyetracking unabhängig vom Labor zugänglich zu machen (Papoutsaki 2018, Abstract). Jedoch hat eine Laborumgebung den Vorteil, dass viele Einflussfaktoren konstant bleiben. Alle Teilnehmer hatten

- den selben Standort,
- den selben Computer mit gleicher Webcam,
- den selben Browser mit selben Einstellungen,
- die selbe Sitzausrichtung,
- die selbe Anweisung,
- die selben Lichtverhältnisse.

Die Lichtverhältnisse wurden versucht konstant zu halten, indem auch bei Tageslicht die Deckenbeleuchtung eingeschaltet war und alle Probanden zu einer ähnlichen Uhrzeit (zwischen 10 und 13 Uhr) die Studie durchführten.

Dass man so viele Faktoren konstant lassen kann, erleichtert spätere Analysen. Deshalb wurde zusätzlich versucht, den Ablauf der Studie in der Ausarbeitungsphase so genau wie möglich festzulegen. Zum methodischen Vorgehen wurde schriftlich fixiert:

1. Versuchsleiter (V) bereitet beide Systeme vor. Insbesondere wird der Browser neu gestartet, um zwischengespeicherte Trainingsdaten von WebGazer zu beseitigen.
2. V bittet Probanden (P) herein.
3. V erklärt das Vorhaben (vgl. unten Studienleitfaden).
4. P führt Studie allein durch, während V für Fragen zur Verfügung steht. Fragen des Probanden werden schriftlich notiert.
5. P beendet Studie. P kann jetzt noch eine E-Mailadresse für Kontakt angeben, falls er Interesse an der Auswertung hat.
6. V bedankt sich und verabschiedet P.

Um eine gewisse Struktur bei der Erklärung der Studie zu haben, wurde für drittens zusätzlich ein Studienleitfaden entworfen. Das Folgende beschreibt die Gesprächsführung während der Studie.

1. Begrüßung
Hallo, danke, dass du zu dieser kleinen Studie gekommen bist.
2. Vorhaben
Wir führen eine kleine Eyetracking-Studie durch. Dafür kannst du dich schon einmal vor diesen Computer setzen und ein paar Angaben zu deiner Person ausfüllen.

3. Kalibrierung

Zuerst werden wir das System kalibrieren. Dafür musst du auf Kreise an verschiedenen Orten klicken, bis sie die Farbe auf grün gewechselt haben. Anschließend erscheint ein Kreis in der Mitte des Bildschirms, der immer kleiner wird. Versuche immer den Mittelpunkt dieses Kreises anzusehen. Du kannst das ganze an einem kurzen Beispiel ausprobieren. (Proband testet 4 Punkte.) Nachher werden es 26 Punkte sein, die du anklicken musst.

4. Aufgabenstellung

Nach der Kalibrierung wird dir die Aufgabe und das weitere Vorgehen beschrieben. Die Aufgabenstellung wird auf Englisch sein. Wenn etwas unverständlich ist, dann kannst du mich gerne zwischendurch fragen, aber bitte schaue mich dabei nicht an, sondern lasse die Augen auf dem Monitor.

5. Studienstart | Versuchsleiter startet WebGazer

Bitte platziere deinen Kopf so, dass er im (grünen) Rechteck ist und du bequem sitzen kannst. Nach dem Klicken auf Start beginnt die Studie mit der Kalibrierung.

6. Studienende | Versuchsleiter schließt Browser-Fenster

Wenn du magst, kannst du deine E-Mailadresse angeben, dann würde ich dich kontaktieren, sobald Ergebnisse vorliegen.

An diesen Leitfaden habe ich mich bis auf zwei kleine Änderungen gehalten. Die erste Änderung ist, dass ich bei der Aufgabenstellung vergessen habe zu erwähnen, dass der Aufgabentext auf Englisch sein wird. Einige Teilnehmer waren davon tatsächlich kurz irritiert. Die zweite Änderung musste ich vornehmen, weil ich vergessen hatte, dass auch die Hardware kalibriert werden muss. Deshalb habe ich nach der Aufgabenstellung mit den Probanden die Hardware kalibriert (s. u.) und anschließend die Studie gestartet.

4.1.3 Angaben zur Person

Ursprünglich war geplant, zusätzlich Daten über die Person mithilfe eines Fragebogens aufzunehmen. Mit Blick auf die Veröffentlichung des Experiments schien es aber angeraten, die Datenaufnahme direkt in die Webapplikation mit zu integrieren. Deshalb wurde eine Formulärseite für diese Angaben entwickelt (vgl. Abbildung 19).

Folgende Daten über die Person werden gesammelt²²:

- Alter und Geschlecht wurden für demografische Unterscheidungen erhoben.
- Die Anzahl der Stunden, die pro Tag am Computer verbracht werden, könnte Auskunft darüber geben, wie viel jemand gewohnt ist, am Bildschirm zu lesen bzw. sich mit Grafiken zu informieren.
- Kontaktlinsen, Brille, Sehschwäche
Für alle Teilnehmer gehörten diese Fragen zusammen. Wer eine Brille trug, gab auch an eine Sehschwäche zu haben. Wie gut jemand sehen kann, beeinflusst den Umgang

²²Bei der Auswahl habe ich mich orientiert an einer Studie mit WebGazer, die von den Entwicklern selber durchgeführt wurde (Papoutsaki 2018, Kapitel 5), (Papoutsaki, Gokaslan u. a. 2018).

The screenshot shows a web-based survey titled 'Adsata'. The left side contains three columns of questions:

- Personal:** Includes fields for age (dropdown), hours of computer use (dropdown), wearing glasses (radio buttons Yes/No), wearing contact lenses (radio buttons Yes/No), gender (radio buttons Male/Female), and weak vision (radio buttons Yes/No).
- Information for face recognition:** Includes dropdowns for eye color (None) and skin color (None), and a text input for 'Something special about your face (e.g. wearing hat, beard)'.
- Information about the environment:** Includes dropdowns for monitor size (In寸), distance to monitor (approximate), input device (Mouse/Trackpad), and ambient light (dropdown).

A large green 'Start Study' button is at the bottom right of the form area. To the right of the form, under the heading 'Beispiel-Objekt in der Datenbank', is a JSON representation of the data collected from this form:

```

{
  "metadata": {
    "age": "26",
    "hours_of_computer_use": "8",
    "glasses": "No",
    "contact_lenses": "No",
    "gender": "male",
    "weak_vision": "yes",
    "eye_color": "Brown",
    "skin_color": "Middle",
    "special_face_features": "light beard",
    "size_monitor": "15",
    "distance_monitor": "40",
    "input_device": "trackpad",
    "ambient_light": "Middle"
  }
}

```

Abbildung 19: Formular für die Angaben zur Person (und Umgebung). Auf der rechten Seite sieht man ein zugehöriges Datenbankobjekt.

mit dem visuellen Reiz. Bei Brillenträgern kommt zusätzlich ins Spiel, ob die Brille einen Einfluss auf die Gesichtserkennung und/oder auf die WebGazer-Daten hat. Da diese Frage entscheidend sein könnte, wäre eine differenzierte Frage nach der Brillenart sinnvoll.

- Die Augenfarbe und Hautfarbe wurde abgefragt, weil diese möglicherweise Einfluss auf die Gesichtserkennung und die WebGazer-Daten haben (Papoutsaki, Gokaslan u. a. 2018).
- Bei den besonderen Angaben zum Gesicht, gab es Einträge zum Tragen von Kopfbedeckungen und zum Bartwuchs.

Die rechte Seite des Formulars musste niemand ausfüllen, da die im Labor für alle gleich waren. Der Prozessor des Computers ist ein Intel(R) Core(TM) i7-3770 CPU @ 3,4 GHz, das Betriebssystem ist Microsoft Windows 10 Pro und die Bildschirmgröße beträgt 23 Zoll. Die Probanden saßen zwischen 50 und 60 Zentimeter vom Monitor entfernt²³, verwendeten alle eine Maus und hatten helle Lichtverhältnisse.

4.1.4 Kalibrierung

Da in der Studie zwei Systeme für Eyetracking verwendet werden, sind auch zwei Kalibrierungsschritte notwendig. Zuerst wird die Hardware richtig eingestellt und im weiteren Studienablauf dann WebGazer kalibriert.

²³Die Hardware gibt für jeden Datenpunkt die Entfernung vom Monitor des jeweiligen Probanden an.

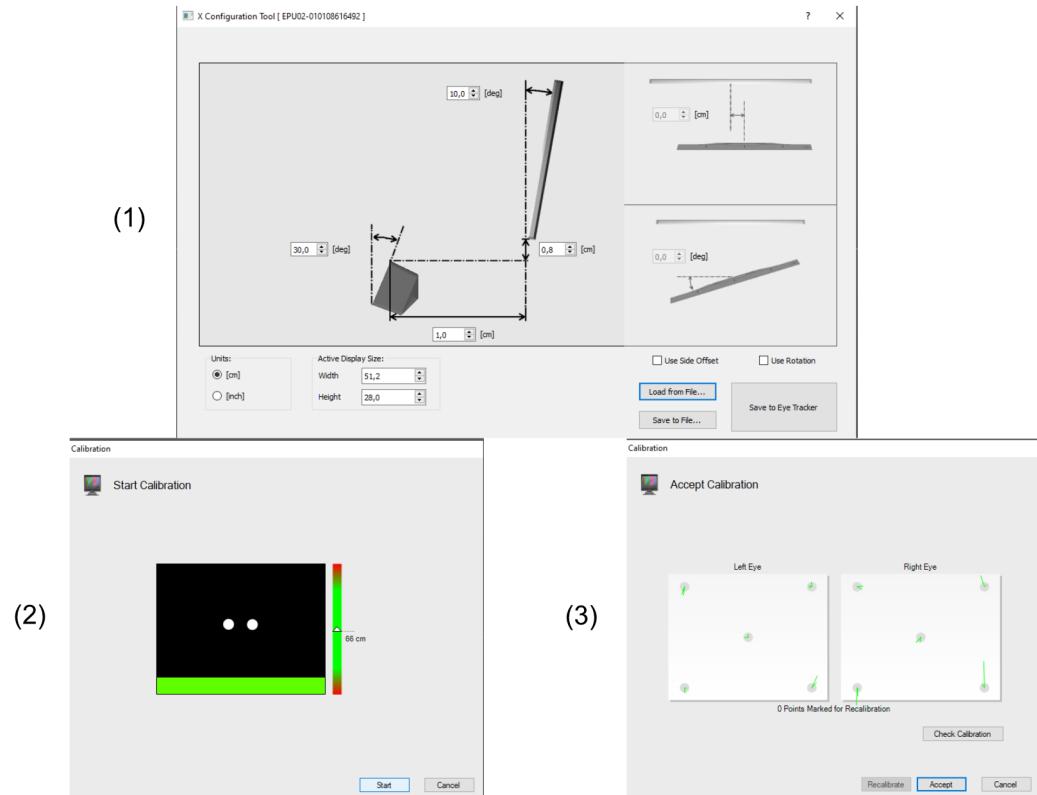


Abbildung 20: Verschiedene Fenster der Tobii-Pro-Software. (1) zeigt das Optionsfenster nach dem Laden der Konfigurationsdatei, in dem die Ausrichtung der Hardware spezifiziert werden kann. (2) zeigt die Ansicht vor der Kalibrierung des Tobii-Pro-Systems. Die beiden Kreise in der Mitte signalisieren, dass die Augen des Probanden erkannt wurden und die rechte Anzeige gibt an, ob der Proband eine geeignete Entfernung hat. (3) zeigt die Qualität der Kalibrierung des Tobii-Pro-Systems. Anhand dieser Visualisierung kann eingeschätzt werden, ob der Kalibrierungsschritt wiederholt werden muss.



Abbildung 21: Kalibrierungsschritt von WebGazer. Links zeigt den Ablauf der Kalibrierung (1-2) und rechts gibt eine Übersicht über alle Positionen der Zielkreise, wobei die Koordinaten in Prozent und die Reihenfolge in den eckigen Klammern angegeben sind.

Um das Tobii-Pro System optimal auszurichten, muss jedes mal, wenn die Software geladen wird (also am Anfang eines Versuchstages) eine Konfigurationsdatei geladen werden (siehe Abb. 20, (1)). Diese teilt dem System mit, in welchem Winkel der Bildschirm und das Tobii-Pro-System ausgerichtet sind und welche Abstände die Eyetracking-Hardware vom Bildschirm in der Höhe und in der Entfernung hat. Bevor die Aufnahme mit der Hardware begonnen wird, startet der Versuchsleiter die Kalibrierung. Angezeigt wird ein roter Kreis mit einem schwarzen Punkt in der Mitte. Dieser bewegt sich über den Bildschirm und muss vom Versuchsteilnehmer mit den Augen verfolgt werden. Mehrere Teilnehmer fragten den Versuchsleiter, ob sie dabei den Kopf bewegen dürfen, worauf die Antwort war, sie können sich ganz natürlich verhalten. Daraufhin waren kleine Kopfbewegungen bei den Teilnehmern zu verzeichnen. Der rote Punkt bewegt sich zu den vier Ecken des Bildschirms sowie zur Mitte, wo er jeweils kurz anhält und kleiner wird. Dieser Ablauf soll dem Teilnehmer dabei helfen, die einzelnen Zielpunkte zu fixieren. Im Anschluss an diesen Kalibrierungsschritt erhält der Versuchsleiter eine Visualisierung der Güte der einzelnen Zielpunkte (vgl. Abb. 20, (3)) und kann gegebenenfalls einzelne oder alle Punkte wiederholen lassen. Im Versuch gab es hier zwischen den Teilnehmern einige Unterschiede. Die Kalibrierung wurde, wenn nötig, wiederholt, aber nach dreimaliger Wiederholung wurde das Ergebnis akzeptiert. Nach der zweiten Wiederholung wurde zudem ein von der Software bereitgestellter Test durchgeführt, wie gut einzelne Punkte auf dem Bildschirm fixiert werden können. Das sollte vor allem den Teilnehmern dabei helfen, zu verstehen, warum eine erneute Kalibrierung nötig ist und dem Versuchsleiter zeigen, wie stark die Abweichungen sind. Zusammenfassend lässt sich sagen, dass selbst bei einer mittelmäßigen Güte des Kalibrierungsschrittes noch eine deutlich höhere Genauigkeit erzielt wurde, als der Versuchsleiter es von den WebGazer-Daten gewöhnt war. Diese Einschätzung trug auch dazu bei, den Kalibrierungsschritt nach dem dritten Mal nicht erneut zu wiederholen, selbst als es bei zwei Teilnehmern wahrscheinlich geboten gewesen war.

Nachdem die Kalibrierung der Hardware durch den Teilnehmer abgeschlossen ist, wird die Studie gestartet und im Browser (Firefox) die Webcam gestartet. Das Gesicht des Teilneh-

mers sollte dann im grünen Rechteck sein, ohne dass er sich zu viel bewegen darf, weil sonst die Hardwarekalibrierung verloren geht. Um hier einen guten Kompromiss zu finden, wurde vor der Studie mit der Sitzhöhe und dem Abstand zum Bildschirm experimentiert, um eine durchschnittliche Einstellung zu finden. Bei großen Personen reichte der Kopf dennoch über das grüne Rechteck hinaus. Für die Gesichtserkennung war das aber kein Problem, die bei jedem Teilnehmer ausnahmslos und sofort funktionierte. Bei der Kalibrierung muss der Teilnehmer fünfmal auf einen gelben Kreis klicken, bis er sich grün verfärbt hat (vgl. Abb. 21, (1),(1a))²⁴. Der Kreis bewegt sich dann in einer Animation zur nächsten Position (1b), wo sich das Prozedere wiederholt. Es gibt insgesamt 26 solcher Kreise, die an neun Stellen des Bildschirms lokalisiert sein können. Der Ablauf folgt einem festen Schema, dass darauf ausgelegt ist, möglichst lange Wege zwischen den Punkten zu generieren (vgl. Abb. 21, (3)). Die äußeren Ecken des Bildschirms werden dabei favorisiert, um die volle Breite und Höhe des Bildschirms auszuloten. Nach den 26 Positionen erscheint in der Mitte des Bildschirms ein großer Kreis (der Radius ist 100 Pixel), der über 16 Sekunden lang immer kleiner wird und zum Schluss nur noch als ein Pixel auf dem Bildschirm zu erkennen ist (2, 2a, 2b). Der Proband muss den Kreis über die gesamten 16 Sekunden fixieren. Die kleiner werdende Animation soll ihm dabei helfen, die Fixierung aufrecht zu halten. Die von WebGazer generierten Daten über diesen Zeitraum werden mit einem gesonderten Label (*errorRate*) versehen und dienen dazu, eine Fehlerrate zu berechnen. Dazu wird der Mittelwert aller euklidischen Distanzen zum Mittelpunkt des Bildschirms (Ort des letzten Pixels vom kleiner werdenden Kreis) berechnet. Diese Berechnung erfolgt clientseitig im Browser, sodass der Wert direkt nach der Kalibrierung zur Verfügung steht. So kann gegebenenfalls die Kalibrierung durch weitere Punkte verlängert oder wiederholt werden, falls der Wert einen bestimmten Grenzwert unterschreitet. Da allerdings zum Zeitpunkt der Studie keine Kenntnisse über die Qualität eines solchen Wertes vorlagen und auch keine Vergleichswerte vorhanden sind, wurde hier der Wert lediglich gespeichert. In der Analyse gibt dieser Wert zusätzliche Auskunft über die Session.

Bevor die Studie begann, wurde das ganze Setting mit zwei Versuchspersonen ausprobiert. Im Anschluss stellte sich heraus, dass die Fehlerrate sehr hoch war (258,1 und 1216,6). Deshalb wurden die Kalibrierungspunkte von 13 auf 26 erhöht. Ein weiterer Test ergab, dass die Fehlerrate dann bei ca. 100 bis 130 Pixel lag. Noch mehr Punkte ergaben keine wesentlichen Veränderungen mehr, weshalb diese Anzahl letztendlich verwendet wurde. Ob auch weniger Punkte ausreichen oder eine andere (vielleicht zufällige) Verteilung auf dem Bildschirm den Wert verbessern, bliebe zu testen. Generell ist wenig über den Kalibrierungsschritt bei WebGazer bekannt (Semmelmann und Weigelt 2018, S. 461).

Aus den beiden Testläufen ergab sich zudem, dass der Kalibrierungsschritt von WebGazer nicht gleich verstanden wurde. Insbesondere der letzte über 16 Sekunden kleiner werdende Punkt dauerte für die Teilnehmer sichtlich zu lange. Um sie deshalb besser auf diese Situation vorzubereiten wurde in der Studie vor dem Start jeweils eine kleine Kalibrierung mit vier Punkten ausprobiert und der Ablauf mit dem letzten Punkt erläutert²⁵. Diese Vorbereitung half, einen flüssigen Ablauf in der Studie zu erreichen.

²⁴WebGazer kalibriert sich selbst über Mausereignisse (Papoutsaki 2018, 24 f.).

²⁵<http://dschr.de/CalibrationTest.html>, abgerufen am 02.12.2020.

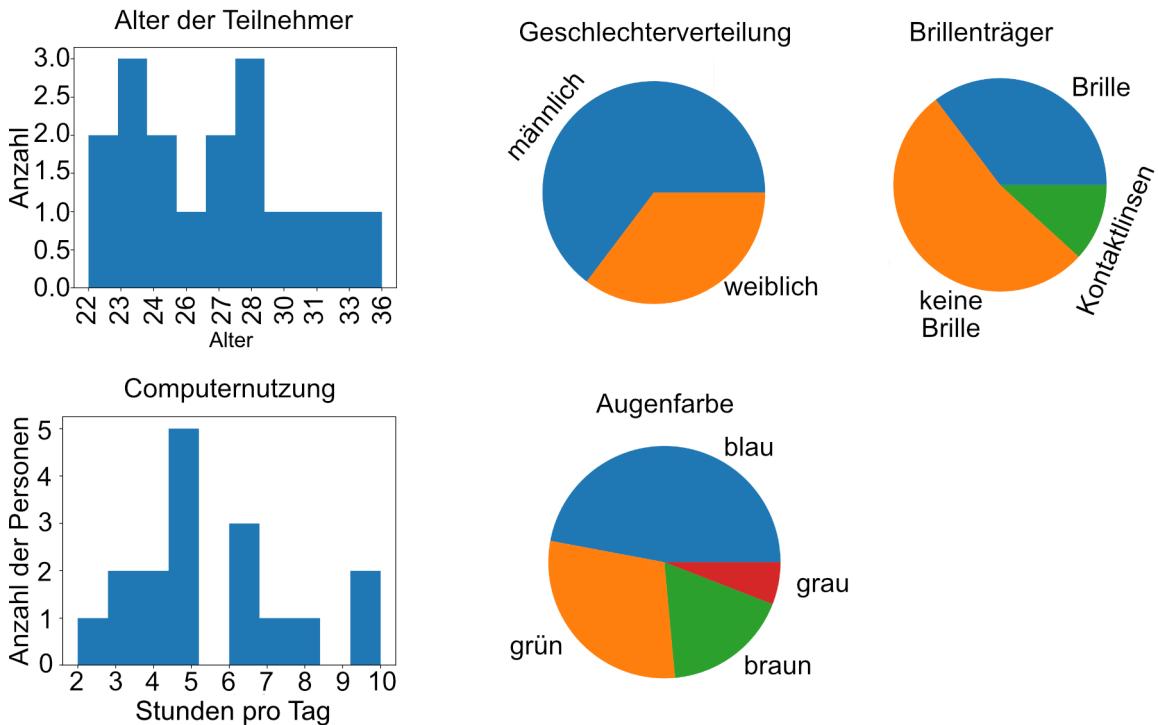


Abbildung 22: Auswertung der allgemeinen Angaben zur Person.

4.1.5 Studie

An die Kalibrierung von WebGazer schloss sich direkt die Studie mit den drei oben beschriebenen Stimuli an. Nachdem die Multiple-Choice-Frage beantwortet war, beendete der Versuchsleiter die Eyetracking-Hardware und verabschiedete den Probanden.

4.2 Daten

In diesem Abschnitt werden die erhobenen Daten vorgestellt. Insgesamt haben 17 Probanden mitgemacht, von denen 16 Datensätze verwendet werden können. Bei einem Probanden hat der Versuchsleiter vergessen, auf den Aufnahmebutton für das Tobii-Pro-System zu klicken, weshalb keine Hardwaredaten vorliegen.

4.2.1 Allgemeine Angaben

Um zu verstehen, aus welchen Teilnehmern der Datensatz zusammengesetzt ist, werden hier überblicksartig die verschiedenen Angaben zur Person vorgestellt (vgl. Abb. 22).

- Alter
Das Alter der Teilnehmer reichte von 22 bis 36 Jahre.
- Geschlecht
Der Anteil der Frauen betrug 6 gegenüber 11 Männern.

- Sehhilfen
6 Personen trugen eine Brille und weitere 2 Personen Kontaktlinsen. (Die Frage der Sehschwäche scheint kein gutes Kriterium zu sein, da sie von allen Brillen- und Kontaktlinsenträgern mit Yes beantwortet wurden. Ursprünglich war damit eine Sehschwäche gemeint, die über das gewöhnliche Maß hinausgeht.)
- Computernutzung pro Tag (in Stunden)
Die meisten Teilnehmer geben an, 5 Stunden pro Tag am Computer zu verbringen, was etwa auch der Durchschnitt ist (≈ 5.53 Stunden).
- Augenfarbe
Knapp die Hälfte der Teilnehmer gab an, blaue Augen zu haben. Die anderen Augenfarben verteilen sich auf grün (5), braun (3) und grau (1).
- Spezielle Angaben zum Gesicht
Sechs Teilnehmer geben an, einen Bart zu tragen und zwei hatten einen Hut auf.

Insgesamt ergibt sich ein Datensatz, der aus dem jungen, akademischen Milieu kommt. Der Umgang mit dem Computer ist vertraut. Ein Drittel aller Teilnehmer trug eine Brille, was bei der Datenauswertung eine besondere Rolle spielen wird. Eine ähnliche Rolle könnte die Augen- und Hautfarbe haben. Sie sind daraufhin zu untersuchen, ob sie Einfluss auf die Gesichtserkennung haben. Während die Augenfarbe etwas weiter gestreut ist, geben bei der Hautfarbe alle *Light* an. Nur ein Teilnehmer beantwortete die Frage nicht oder gab bewusst *None* an. Einen weiteren Unterschied könnte ein Bart oder eine Kopfbedeckung machen.

4.2.2 Daten von WebGazer

Von WebGazer erhalten wir die gleiche Datenstruktur, wie in Kapitel 3.1.4 beschrieben: (x, y) Koordinaten und einen Zeitstempel. Der einzige Unterschied ist nur, dass die Daten entsprechend ihres Auftretens in die Objekte *calibrationData*, *readingPageData*, *plotPageData* und *questionPageData* aufgeteilt sind. Insgesamt haben wir 19145 WebGazer-Datenpunkte gesammelt. Davon entfallen die meisten Daten auf die Seite mit der Grafik (11134), die Aufgabenbeschreibung hat 6422 Datenpunkte und die letzte Seite nur noch 1589.²⁶

Während die Multiple-Choice-Frage und der Text (Aufgabenbeschreibung) fast immer dieselbe Menge an Datenpunkten hat, gibt es bei der Grafik starke Schwankungen (vgl. Abb. 23). Es gibt Sessions mit vielen Datenpunkten (2, 4, 14) und Sessions mit wenigen Datenpunkten (6, 9, 11). Die Schwankungen sind damit zu erklären, dass die Seite mit der Grafik kein festes Ziel hat. Der Text auf der vorherigen Seite ist irgendwann gelesen und die Multiple-Choice-Frage schnell beantwortet, aber bei der Grafik gibt es kein klares Ende, wann die Aufgabe erfüllt ist. Es hängt also von der Testperson ab, wann sie glaubt, fertig zu sein.

Durchschnittlich lieferte WebGazer alle 61.1 ms ein Datum. Das entspricht einer ungefähren Samplingrate von 16.4 Hz. Dieser Wert ist realistisch. Holmqvist ordnet dem Spektrum 25-30 Hz die langsamsten Systeme zu, die überhaupt verkauft werden und weist darauf hin, dass nur Webcam-basierte Ansätze langsamer seien (Holmqvist 2011, S. 30). Dennoch haben wir in Testdaten Samplingraten von 20 bis 25 Hz gefunden.²⁷ Die geringe Samplingrate könnte

²⁶Dass die Multiple-Choice-Frage nur sehr wenige Datenpunkte hat, ist ein Indiz dafür, dass die Frage leicht und damit schnell beantwortet werden konnte.

²⁷So liegt z. B. die Samplingrate beim Datensatz aus Kapitel 3.2 bei ca. 23 Hz.

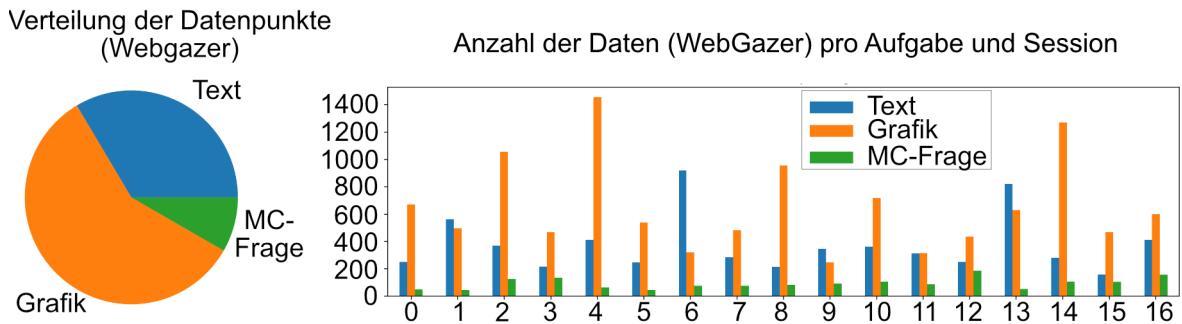


Abbildung 23: Verteilung der Daten von WebGazer auf die einzelnen Stimuli.

dadurch zustande gekommen sein, dass der Computer durch die parallele Aufnahme mit der Hardware so ausgelastet war, dass nicht genügend Ressourcen zur Verfügung standen, um die Berechnungen von WebGazer schneller durchzuführen. Von der Samplingrate hängt ab, wie viele WebGazer-Daten aufgenommen werden. Umso geringer sie ist, umso mehr Hardwaredaten steht ein WebGazer-Datenpunkt gegenüber.

4.2.3 Daten von Tobii-Pro

Das Tobii-Pro-System liefert insgesamt 86 Dimensionen pro Datenpunkt. Um einen kleinen Überblick über die breite der Informationen zu geben, seien hier einige Gebiete zusammengefasst:

- System, welches die Daten aufgenommen hat.
- Projekt, in dem diese Daten aufgenommen werden.
- Medien, die während der Aufnahme gezeigt werden und welche Maße und Positionen sie haben.
- Ereignisse durch Maus, Tastatur, Tobii-Pro-Software oder Betriebssystem, die während der Aufnahme eintreffen.
- Positionen der Augen, Pupillen und des Kamerasytems sowie die Distanz der Augen zur Kamera.

Auf diese ganzen Daten kann in der vorliegenden Arbeit nicht eingegangen werden. Ihre Aufzählung dient der Illustration, welche Analysemöglichkeiten zusätzlich bestehen. Wir konzentrieren uns auf die Informationen, die mit den WebGazer-Daten verglichen und synchronisiert werden sollen. Das betrifft die zeitliche Komponente und die Koordinaten der Datenpunkte auf dem Bildschirm²⁸

Wie bei den Daten von WebGazer hat das Tobii-Pro-System einen Zeitstempel in Millisekunden, seitdem die Aufnahme gestartet wurde (*RecordingTimestamp*, vgl. Abb. 25). Die

²⁸Diese Koordinaten sind nicht dasselbe wie die Position der Augen und Pupillen sondern bereits das Ergebnis aus diesen Daten, wo die Testperson auf dem Bildschirm hingeschaut hat.

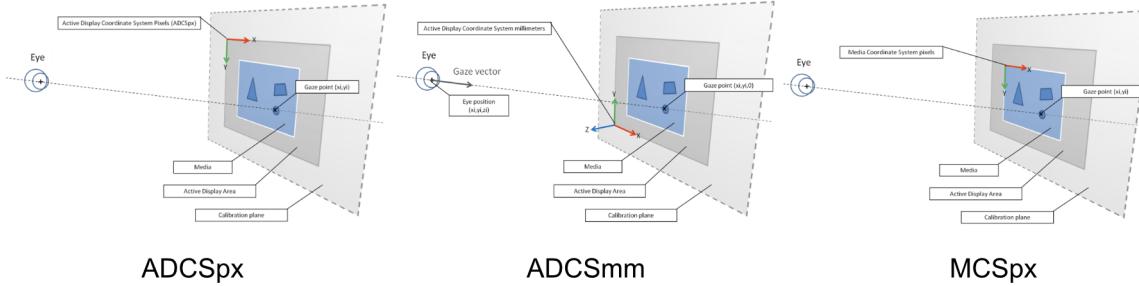


Abbildung 24: Die verschiedenen Koordinatensysteme von Tobii-Pro-Studio im Überblick.

Software speichert zudem für jeden Datenpunkt das Datum (*RecordingDate*) und die lokale Uhrzeit (*LocalTimeStamp*), welche ebenfalls bis auf Millisekunden genau ist. Zusätzlich hat die Hardware eine eigene Zeitmessung, die mit einem ungewissen Wert startet und eine Auflösung in Mikrosekunden hat.²⁹

Das Tobii-Pro-System verwendet drei Koordinatensysteme (vgl. Abb. 24).

1. Active Display Coordinate System pixels (ADCSpx)

Das System ist ein 2D Koordinatensystem, welches auf den Bereich des Bildschirms ausgerichtet ist, welcher vorher als aktiv eingestellt wurde. In unserem Fall ist die aktive Fläche der gesamte Bildschirm, so dass diese Koordinaten vergleichbar mit denen von WebGazer sind.

2. Active Display Coordinate System millimeters (ADCSmm)

Auch dieses Koordinatensystem ist an den aktiven Bereich des Bildschirms ausgerichtet, ist aber drei-dimensional. Aus der Augenposition und der Distanz zum Monitor wird die $(x, y, 0)$ Koordinate berechnet, wobei der z -Wert null ist für den Punkt auf dem Bildschirm. Zu beachten ist, dass sich der Koordinatenursprung unten links befindet.

3. Media Coordinate System pixels (MCSpx)

Dieses Koordinatensystem ist wieder in 2D, aber diesmal an dem dargestellten Medium ausgerichtet. Da das Experiment im Vollbildmodus durchgeführt wurde, gibt es hier keinen Unterschied zu erstens.

Neben den drei Koordinatensystemen differenziert die Eyetracking-Hardware die Koordinate hinsichtlich der beiden Augen. Wir haben also für die Analyse jeweils eine (x, y) Koordinate für jedes Auge sowie eine aus beiden Daten gemittelte Angabe. Letztere ist vergleichbar mit der Koordinate von WebGazer. Die Daten werden bezeichnet als *GazePointX* (*ADCSpx*) und *GazePointY* (*ADCSpx*).

²⁹ Alle Angaben zu den Hardwaredaten sind der Anleitung zum Tobii-Studio entnommen. Diese ist online abrufbar unter: <https://www.tobiipro.com/siteassets/tobii-pro/user-manuals/tobii-pro-studio-user-manual.pdf>, abgerufen am 20.12.2020.

	RecordingTimestamp	RecordingDate	LocalTimeStamp	EyeTrackerTimestamp		
500	4160	21.10.2020	11:27:05.352	2.942952e+09		
501	4169	21.10.2020	11:27:05.361	2.942960e+09		
502	4177	21.10.2020	11:27:05.369	2.942968e+09		
503	4185	21.10.2020	11:27:05.377	2.942977e+09		
504	4194	21.10.2020	11:27:05.386	2.942985e+09		
...		
995	8285	21.10.2020	11:27:09.477	2.947076e+09		
996	8293	21.10.2020	11:27:09.485	2.947085e+09		
997	8301	21.10.2020	11:27:09.494	2.947093e+09	Zeitdimension (Tobii-Pro)	
998	8310	21.10.2020	11:27:09.502	2.947101e+09		
999	8318	21.10.2020	11:27:09.510	2.947110e+09		

timestamp	x	y	RecordingTimestamp	GazePointX (ADCSpX)	GazePointY (ADCSpX)	GazeEventType
61647	1797.666919	939.601442		79090	948.0	555.0
61731	1809.596444	891.033840		79099	979.0	557.0
61782	1744.398223	916.114408		79107	971.0	531.0
61849	1652.243051	862.602094		79115	898.0	554.0
61923	1626.571109	826.088778		79124	854.0	376.0
...
77390	648.333481	301.618820		95080	NaN	NaN
77440	679.871092	367.925301		95089	939.0	593.0
77524	734.223561	472.242546		95097	NaN	NaN
77577	658.464978	468.002587		95105	947.0	483.0
77640	552.095417	401.868675		95114	990.0	596.0

WebGazer	Tobii-Pro	Label

Abbildung 25: Die Abbildung zeigt die verschiedenen Daten, die für die Zusammenführung von WebGazer- und Hardware-Daten benötigt werden. Oben sind die Zeitangaben der Hardwaredaten dargestellt, die für die Synchronisierung wichtig sind. Unten sind die Koordinaten von WebGazer und Tobii-Pro und dessen Label abgebildet.

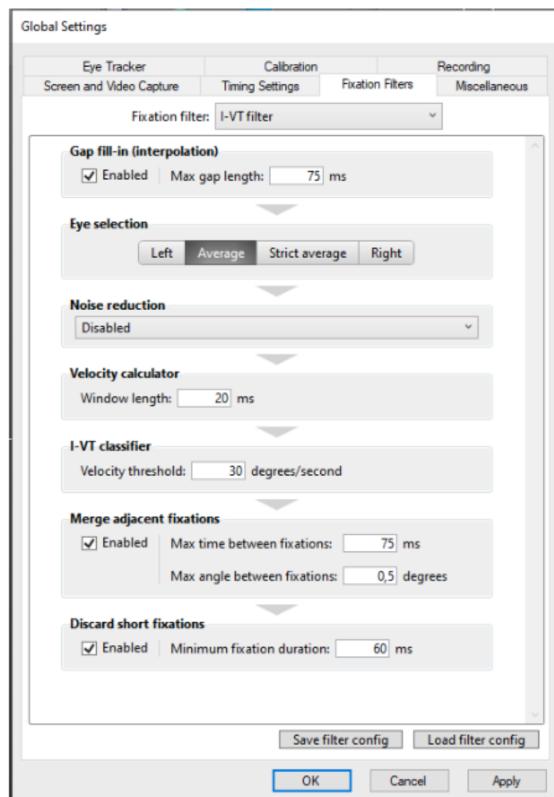


Abbildung 26: Fenster von Tobii-Pro-Studio zum Einstellen des Fixationsfilters. Die eingestellten Werte sind Vorgaben von der Software, die beibehalten wurden.

	timestamp	x	y	RecordingDate	LocalTimeStamp	GazePointX (ADCSpx)	GazePointY (ADCSpx)	GazeEventType	
0	125382	805.724813	598.797895	0	29.10.2020	10:55:19.501	936.0	508.0	Unclassified
1	125442	830.150531	575.433364	1	29.10.2020	10:55:19.510	928.0	504.0	Fixation
2	125527	853.116613	571.833358	2	29.10.2020	10:55:19.518	938.0	517.0	Fixation
3	125578	840.831675	574.131049	3	29.10.2020	10:55:19.526	916.0	548.0	Fixation
4	125628	841.620069	571.197856	4	29.10.2020	10:55:19.535	927.0	530.0	Fixation

WebGazer

Tobii-Pro

Abbildung 27: Diese Datenstruktur wird für die Synchronisierung verwendet.

Um aus den Hardwaredaten die gleiche Datenstruktur wie bei den WebGazer-Daten zu bekommen, müsste man sie nach *RecordingTimestamp*, *GazePointX (ADCSpx)* und *GazePointY (ADCSpx)* filtern (vgl. Abb. 25). Zusätzlich wollen wir noch die Klassifizierung der einzelnen Datenpunkte durch die Hardware verwenden. Beim Export der Daten kann ein Fixations-Filter eingestellt werden, der die Daten entweder als *Fixation*, *Saccade* oder *Unclassified* bezeichnet. Aus mangelnder Erfahrung wurde die Voreinstellung beibehalten. Ausgewählt war der I-VT-Filter (Salvucci und Goldberg 2000; Andersson u. a. 2016) mit festgelegten Parametern wie in Abbildung 26.

Im nächsten Abschnitt wird gezeigt, wie die WebGazer- und Hardware-Daten bearbeitet werden müssen, um die Daten der beiden Systeme zu verbinden. Dafür wird ein externer Zeitstempel verwendet, der nach dem Kalibrierungsschritt von WebGazer gespeichert wurde.

4.2.4 Vorverarbeitung

Von den Hardwaredaten brauchen wir nur eine kleine Auswahl. Die Daten werden auf die Attribute *RecordingDate*, *LocalTimeStamp*, *GazePointX (ADCSpx)*, *GazePointY (ADCSpx)* und *GazeEventType* projiziert. Von WebGazer werden alle Daten verwendet (vgl. Abb. 27). Um die Daten von WebGazer und Tobii-Pro zusammenführen zu können, wurde in jeder Session ein von beiden Systemen unabhängiger Zeitstempel gespeichert. Dieser ist ein Unix-Zeitstempel, der nach dem Kalibrierungsschritt von WebGazer aufgenommen und in dem Datenbankeintrag *timestampForSync* gespeichert wurde. Der Unix-Zeitstempel ist die Zeitangabe in Sekunden seit dem 01. Januar 1970 00:00 Uhr UTC. Dieser Nullpunkt wird als *Unix Epoch* bezeichnet.³⁰ Der Vorteil dieser Zeitmessung ist, dass sie unabhängig von der lokalen Zeitmessung immer gleiche Werte hat. Das erlaubt es, verschiedene Systeme miteinander zu synchronisieren. Die Hardwaredaten haben einen solchen Zeitstempel nicht. Deshalb muss dieser in einem ersten Vorverarbeitungsschritt aus den vorhandenen Zeitangaben berechnet werden. Dafür werden die Angaben *RecordingDate* und *LocalTimeStamp* verwendet (vgl. Abb. 28, (1)). Diese beiden Angaben werden in ein Objekt (*DateTime*) aggregiert (2) und anschließend unter Angabe der Zeitzone *Europa/Berlin* in ein Unix-Zeitstempel umgewandelt (3). Da der Zeitstempel *timestampForSync* in Millisekunden gespeichert ist, müssen nun noch alle Werte mit 1000 multipliziert werden.

³⁰Vgl. <https://www.unixtimestamp.com/>, abgerufen am 06.12.2020

	RecordingDate	LocalTimeStamp		RecordingDate	LocalTimeStamp	Date	Time
(1)	0 29.10.2020	10:55:19.501		0 29.10.2020	10:55:19.501	29.10.2020	10:55:19.501
	1 29.10.2020	10:55:19.510		1 29.10.2020	10:55:19.510	29.10.2020	10:55:19.510
	2 29.10.2020	10:55:19.518		2 29.10.2020	10:55:19.518	29.10.2020	10:55:19.518
	3 29.10.2020	10:55:19.526		3 29.10.2020	10:55:19.526	29.10.2020	10:55:19.526
	4 29.10.2020	10:55:19.535		4 29.10.2020	10:55:19.535	29.10.2020	10:55:19.535

	RecordingDate	LocalTimeStamp	Date	Time	timestamp
(3)	0 29.10.2020	10:55:19.501	29.10.2020	10:55:19.501	1.603965e+12
	1 29.10.2020	10:55:19.510	29.10.2020	10:55:19.510	1.603965e+12
	2 29.10.2020	10:55:19.518	29.10.2020	10:55:19.518	1.603965e+12
	3 29.10.2020	10:55:19.526	29.10.2020	10:55:19.526	1.603965e+12
	4 29.10.2020	10:55:19.535	29.10.2020	10:55:19.535	1.603965e+12

Abbildung 28: Die Abbildung zeigt den Weg, wie aus den gegebenen Zeitangaben des Tobii-Pro-Systems ein Unix-Zeitstempel entsteht. Zuerst werden die Datenfelder aus (1) verknüpft (2) und diese dann umgerechnet (3).

In den Hardwaredaten zeigt sich ein weiterer Nachteil, wenn lokale Zeitstempel verwendet werden: Wegen der Zeitumstellung am 25.10.2020 in Deutschland sind beide Zeitmessungen um eine Stunde auseinander. Während der Wert *timestampForSync* die aktuelle Zeit über den Browser speichert, wurde das Tobii-Pro-System nicht umgestellt. Deshalb sind alle lokalen Zeitangaben der Hardwaredaten eine Stunde zu spät. Um das zu korrigieren, muss von den erzeugten *timestamp*-Werten jeweils eine Stunde (60 Minuten * 60 Sekunden * 1000 Millisekunden) subtrahiert werden.

Als Ergebnis haben wir nun in den Hardwaredaten eine Spalte *timestamp*, die um den Zeitpunkt von *timestampForSync* ihre Werte hat.³¹

Bei den WebGazer-Daten muss kein Synchronisierungszeitpunkt gefunden werden, weil schon beim Speichern die Daten den einzelnen Stimuli zugeordnet wurden (vgl. Kapitel 4.2.2). Der erste Datenpunkt im Objekt *readingPageData* ist dementsprechend das erste Datum, welches nach dem Synchronisierungszeitpunkt gesammelt wurde. Dieser Punkt ist zu verbinden mit dem ersten Hardwaredatum, welches direkt nach dem Synchronisierungszeitpunkt aufgenommen wurde (vgl. Abb. 29).

Wir haben also bis jetzt die beiden Datenpunkte herausgearbeitet, bei denen die eigentliche Studie mit dem ersten Stimulus (Aufgabenbeschreibung) beginnt. Bevor die Methode vorgestellt wird, wie ab diesem Zeitpunkt die Daten von WebGazer und Tobii-Pro verbunden werden, wird zunächst überprüft, wie genau der Synchronisierungszeitpunkt ist. Dazu werden die letzten 16 Sekunden vor diesem Zeitpunkt betrachtet. In dieser Zeitspanne war der kleiner werdende Kreis in der Mitte des Bildschirms zu sehen (vgl. Kapitel 4.1.4). Dementsprechend ist zu erwarten, dass sich alle Datenpunkte in der Mitte des Bildschirms konzentrieren.

³¹Die Hardwaredaten beginnen noch vor der Kalibrierung von WebGazer, weshalb also auch Daten vor dem Synchronisierungszeitpunkt vorliegen.

DateTime	timestamp	GazePointX (ADCSpX)	GazePointY (ADCSpX)	GazeEventType	Tobii-Pro
17151	29.10.2020 10:57:41.045	1603961861045.000000	931.000000	513.000000	Fixation
timestampForSync: 1603961861037					
timestamp					
0	125382	805.724813	598.797895		WebGazer

Abbildung 29: Die Abbildung zeigt den ersten gemeinsamen Datenpunkt von Tobii-Pro und WebGazer. Das Beispiel stammt vom ersten Probanden.

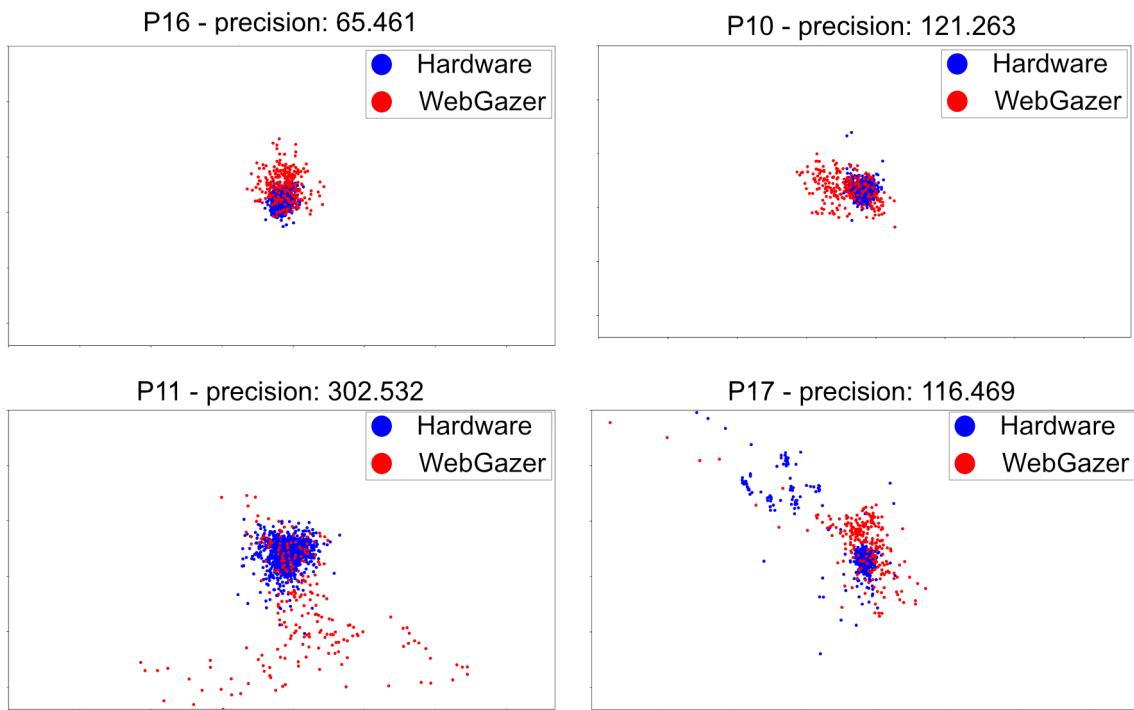


Abbildung 30: Vier unterschiedliche Ergebnisse der letzten 16 Sekunden vor dem Synchronisierungszeitpunkt. Die Teilnehmer fokussierten zu dieser Zeit gerade den kleiner werdenden Kreis am Ende des Kalibrierungsschrittes von WebGazer. Zusätzlich ist ein *precision*-Wert angegeben, der die durchschnittliche euklidische Distanz der WebGazer-Datenpunkte zum Mittelpunkt berechnet.

In Abbildung 30 sieht man das Ergebnis. Die oberen beiden Grafiken (P16 und P10) zeigen gute Ergebnisse mit Verschiebungen nach oben und nach links. Im Allgemeinen aber entsprechen die Darstellungen den Erwartungen. Die Datenpunkte gruppieren sich alle um die Mitte des Bildschirms. Im Datensatz gibt es aber auch zwei Ausnahmen, die in Abbildung 30 unten dargestellt sind. Beim Probanden P11 weichen die WebGazer-Daten nicht nur von der Mitte des Bildschirms ab, sondern unterscheiden sich damit auch signifikant von denen der Hardwaredaten. Letztere sind auch weiter um den Mittelpunkt gestreut als bei anderen Sessions, aber dennoch bleiben sie im Gegensatz zu den WebGazer-Daten um den Mittelpunkt gruppiert. Ein anderes Bild ergibt sich bei der unteren rechten Darstellung. Auch hier weichen die Daten vom Mittelpunkt ab, aber das betrifft WebGazer und Tobii-Pro gleichermaßen. Der Proband scheint hier nicht nur den Kreis fixiert zu haben. Eine Augenbewegung nach oder von oben links ist in beiden Datensätzen zu erkennen. Während WebGazer nur einzelne verstreute Datenpunkte in der oberen linken Ecke hat, bilden sich bei den Hardwaredaten ganze Gruppen von Punkten, was an der höheren Samplingrate liegt.

Um den Synchronisierungszeitpunkt weiter zu validieren, kann man in den Hardwaredaten auch nach Punktwolken um den Mittelpunkt suchen, die eine Fixationsdauer von 16 Sekunden haben. Dafür wird der Mittelpunkt definiert als die Hälfte der Fenstergröße- und breite. Anschließend werden mit zwei Parametern ein Rechteck um den Mittelpunkt definiert, in dem die Datenpunkte liegen dürfen. Die Daten werden nach diesen Maßgaben gefiltert. Beispielhaft wurde das für den Probanden P16³² mit einem Quadrat von 50 Pixel um den Mittelpunkt gemacht und die Ergebnisse in einer Textdatei gespeichert. Dort findet man ab Zeile 10059 über fast 18 Sekunden lang (bis Zeile 12208) Fixationspunkte in dem Quadrat (vgl. Abb. 31, (1)). Die anschließenden Datenpunkte müssten dann den Anfang der Studie markieren. Die Daten in den 10 darauffolgenden Zeilen (2) zeigen eine Sakkade nach oben links, wo der Anfang des Aufgabentextes sich befindet. Daraus folgt, dass der Synchronisierungszeitpunkt zwischen den Zeiten der Zeilen 12208 und 12209 liegen sollte. Der gespeicherte Synchronisierungszeitpunkt (*timestampForSync*) ist allerdings schon bei 10:56:57.434³³, wäre also bereits in Zeile 12148 erreicht. Das entspricht etwa einer halben Sekunde, bis die Sakkade beginnt. Diese halbe Sekunde erklärt sich daraus, dass die Teilnehmer noch etwas länger auf die Mitte des Bildschirms schauen, bis sie realisiert haben, dass der Punkt verschwunden ist. Somit ist der Synchronisierungszeitpunkt richtig, aber die ersten gesammelten Daten konzentrieren sich um den Mittelpunkt, bis der Teilnehmer seinen Blick von dieser Stelle löst.

4.3 Synchronisierung der Daten von WebGazer und Tobii-Pro

Im vorigen Abschnitt wurde gezeigt, dass der gespeicherte Synchronisierungszeitpunkt plausibel ist. Er wird nun verwendet, um die Daten von WebGazer und Tobii-Pro zu verbinden. Beispielhaft führe ich das Verfahren hier für den ersten Probanden vor.³⁴

Zuerst wird die Dauer der jeweiligen Session berechnet. Diese ergibt sich aus der Zeitdifferenz zwischen dem ersten WebGazer-Datenpunkt von dem ersten Stimulus (im Objekt *readingPageData*) und dem letzten Datenpunkt des dritten Stimulus (im Objekt *questionPageData*, vgl.

³²Siehe auch im Jupyter-Notebook *findCalibrationData*.

³³Es sei daran erinnert, dass die Daten um eine Stunde verschoben sind, weil die Systeme der Hardware noch nicht an die Zeitumstellung angepasst waren.

³⁴Der Quellcode für diese Session kann im Jupyter-Notebook *CombineData* nachgelesen werden. Der Quellcode für alle Sessions befindet sich in der gleichnamigen Python-Datei.

RecordingTimestamp	LocalTimeStamp	GazePointX (ADCSpX)	GazePointY (ADCSpX)	GazeEventType
10059	82744	11:56:40.076	1004.0	597.0
10060	82752	11:56:40.085	1015.0	592.0
10061	82761	11:56:40.093	999.0	611.0
10062	82769	11:56:40.101	1008.0	586.0
10063	82777	11:56:40.110	1007.0	606.0
...
12204	100575	11:56:57.908	1024.0	575.0
12205	100584	11:56:57.916	1016.0	574.0
12206	100592	11:56:57.924	1017.0	555.0
12207	100600	11:56:57.933	1000.0	581.0
12208	100609	11:56:57.941	1012.0	528.0

(1)

RecordingTimestamp	LocalTimeStamp	GazePointX (ADCSpX)	GazePointY (ADCSpX)	GazeEventType
12209	100617	11:56:57.949	995.0	558.0
12210	100625	11:56:57.958	1001.0	530.0
12211	100634	11:56:57.966	1001.0	540.0
12212	100642	11:56:57.974	994.0	507.0
12213	100650	11:56:57.983	958.0	483.0
12214	100659	11:56:57.991	935.0	470.0
12215	100667	11:56:57.999	843.0	358.0
12216	100675	11:56:58.008	756.0	239.0
12217	100684	11:56:58.016	695.0	271.0
12218	100692	11:56:58.024	723.0	249.0
12219	100700	11:56:58.033	727.0	255.0

(2)

Abbildung 31: Die Abbildung zeigt, dass 16 Sekunden vor dem Synchronisierungszeitpunkt vornehmlich Fixationen um den Mittelpunkt des Bildschirms in den Hardwaredaten zu finden sind (1) und sich daran eine Sakkade anschließt (2).

	timestamp	x	y		timestamp	GazePointX (ADCSpX)	GazePointY (ADCSpX)	GazeEventType	
0	125382	805.724813	598.797895		17150	1.603962e+12	932.0	494.0	Fixation
1	125442	830.150531	575.433364		17151	1.603962e+12	931.0	513.0	Fixation
2	125527	853.116613	571.833358		17152	1.603962e+12	931.0	527.0	Fixation
3	125578	840.831675	574.131049		17153	1.603962e+12	970.0	551.0	Fixation
4	125628	841.620069	571.197856		17154	1.603962e+12	984.0	553.0	Fixation
...	
1098	194304	992.337366	583.112148		25449	1.603962e+12	889.0	490.0	Fixation
1099	194358	939.116101	544.840858		25450	1.603962e+12	890.0	488.0	Saccade
1100	194409	925.749212	530.538707		25451	1.603962e+12	903.0	457.0	Unclassified
1101	194490	928.691400	529.659512		25452	1.603962e+12	898.0	492.0	Saccade
1102	194542	1021.360609	556.513418		25453	1.603962e+12	845.0	557.0	Saccade
duration = 194542 - 125382 = 69160				> timestampForSync	< timestampForSync + duration				

WebGazer-Daten

Tobii-Pro-Daten

Abbildung 32: Die Abbildung zeigt das Vorgehen, die Hardwaredaten anhand der Dauer der Studie zu filtern.

Abb. 32)³⁵. Die Session des ersten Teilnehmers dauerte 69,16 Sekunden. Nun können die Hardwaredaten nach dem Zeitraum gefiltert werden, der zwischen dem Synchronisierungszeitpunkt (*timestampForSync*) und der addierten Dauer liegt (vgl. Abb. 32), rechts).

Auf beiden Datensätzen (WebGazer und Tobii-Pro) wird jeweils eine neue Spalte *timeDiff* eingeführt. Diese beschreibt die Zeitdifferenz des jeweiligen Datenpunktes zum Beginn der Session (also die Zeitdifferenz zum ersten Datenpunkt). Um das zu erreichen wird vom aktuellen Zeitstempel (*timestamp*) der erste Zeitstempel der Session abgezogen. Dieser fungiert als Nullpunkt, von dem aus die Studie dann verläuft (vgl. Abb. 33, oben).

Mithilfe dieser Spalte werden die beiden Datensätze jetzt über einen *Full Outer-Join* miteinander verbunden und nach der *timeDiff* sortiert. Alle Daten beider Datensätze bleiben so erhalten. Gibt es einen gemeinsamen Wert bei dem Attribute *timeDiff*, werden diese beiden Datenpunkte verknüpft (vgl. Abb. 33, mitte, Spaltennummer 0). Hat nur der Datensatz von Tobii-Pro einen Wert in der Spalte *timeDiff*, werden die WebGazer-Daten mit Nullwerten³⁶ aufgefüllt (Spaltennummern 1 - 7) und andersherum (Spaltennummer 8304).

Da die Samplingrate bei Tobii-Pro höher ist, kommen auf einen Datenpunkt von WebGazer mehrere Daten. (So ist bspw. das WebGazer-Datum in Spaltennummer 8304 umgeben von 17 Tobii-Pro-Daten.) Das Ziel ist, jeder WebGazer-Koordinate eine Hardware-Koordinate und ein Label (*Fixation*, *Saccade* oder *Unclassified*) zuzuordnen. Dazu werden im WebGazer-Datensatz zwei neue Spalten (*before* und *after*) erzeugt, die jeweils die halbierte Zeitdif-

³⁵Zum Ablauf einer Session und zur Struktur der gespeicherten Daten vgl. Kapitel 4.1.1 und 4.2.2

³⁶Der besseren Übersicht wegen wurden die Nullwerte durch Leerzeichen ersetzt.

The diagram illustrates the data flow and mapping between different types of gaze data:

- Left Table (Hardware Points):** Shows a sequence of timestamped hardware data points (x, y coordinates) over time.
- Middle Table (Gaze Events):** Shows a sequence of timestamped gaze events (GP_X, GP_Y) with their corresponding event types (Fixation, Saccade, Unclassified).
- Bottom Table (Combined Data):** Shows a final table where each hardware point is associated with its timestamp, x, y, timeDiff, and the corresponding GazeEvent from the middle table. It also includes columns for 'before' and 'after' time differences relative to the current timestamp.
- Annotations:**
 - A green box highlights the 'timeDiff' column in the first table.
 - A green box highlights the 'timeDiff' column in the middle table.
 - A blue dashed box highlights the 'before' and 'after' columns in the bottom table.
 - Handwritten calculations on the right side show the calculation of time differences: $60 - 30 = 30$ and $60 + 42.5 = 102.5$.

	timestamp	x	y	timeDiff
0	125382	805.724813	598.797895	0
1	125442	830.150531	575.433364	60
2	125527	853.116613	571.833358	145
3	125578	840.831675	574.131049	196
4	125628	841.620069	571.197856	246
...
1098	194304	992.337366	583.112148	68922
1099	194358	939.116101	544.840858	68976
1100	194409	925.749212	530.538707	69027
1101	194490	928.691400	529.659512	69108
1102	194542	1021.360609	556.513418	69160

	timestamp	GP_X	GP_Y	GazeEventType	timeDiff
17150	1.603962e+12	932.0	494.0	Fixation	0.0
17151	1.603962e+12	931.0	513.0	Fixation	8.0
17152	1.603962e+12	931.0	527.0	Fixation	17.0
17153	1.603962e+12	970.0	551.0	Fixation	25.0
17154	1.603962e+12	984.0	553.0	Fixation	33.0
...
25449	1.603962e+12	889.0	490.0	Fixation	69126.0
25450	1.603962e+12	890.0	488.0	Saccade	69134.0
25451	1.603962e+12	903.0	457.0	Unclassified	69142.0
25452	1.603962e+12	898.0	492.0	Saccade	69151.0
25453	1.603962e+12	845.0	557.0	Saccade	69159.0

	x	y	timeDiff	GazePointX (ADCSpX)	GazePointY (ADCSpX)	GazeEventType
0	805.725	598.798	0.0	932	494	Fixation
1			8.0	931	513	Fixation
2			17.0	931	527	Fixation
3			25.0	970	551	Fixation
4			33.0	984	553	Fixation
5			42.0	948	531	Fixation
6			50.0	985	549	Fixation
7			58.0	971	557	Fixation
8304	830.151	575.433	60.0			
8			67.0	973	537	Fixation
9			75.0	930	557	Fixation
10			83.0	976	562	Fixation
11			92.0	948	567	Fixation
12			100.0	934	567	Fixation
13			108.0	954	558	Fixation
14			117.0	955	558	Fixation
15			125.0	967	546	Fixation
16			133.0			Fixation
17			142.0	956	550	Fixation
8305	853.117	571.833	145.0			
18			150.0	964	552	Fixation

	timestamp	x	y	timeDiff	before	after
0	125382	805.724813	598.797895	0	0.0	30.0
1	125442	830.150531	575.433364	60	30.0	42.5
2	125527	853.116613	571.833358	145	42.5	25.5
3	125578	840.831675	574.131049	196	25.5	25.0
4	125628	841.620069	571.197856	246	25.0	35.5

Abbildung 33: Die Abbildung beschreibt den vollständigen Weg der Datenverknüpfung.⁵² Jedem WebGazer-Datenpunkt (oben links) werden mehrere Hardware-Datenpunkte (oben links) zugeordnet (mittig).
 (before) (after)

ferenz zum vorherigen und nächsten Datenpunkt enthält (vgl. Abb. 33, unten). Mithilfe dieser Informationen wird von jedem WebGazer-Datenpunkt ($timestamp, x, y$) aus, diejenige Gruppe von Hardwaredaten ausgewählt, die im zeitlichen Fenster ($timestamp - before, timestamp + after$) liegt (vgl. Abb. 33, blau-gestrichelten Rechtecke). Diese Gruppe von Hardwaredaten muss nun zu einem Datum zusammengefasst werden. Von den x -und y-Werten wird der Durchschnitt berechnet. Bei den Labeln wird dasjenige ausgewählt, welches am Häufigsten auftritt. Treten zwei gleichgroße Klassen auf, dann wird das erste Auftreten ausgewählt.³⁷

Das Ergebnis der ganzen Unternehmung ist ein neuer Datensatz für jede Session, wie er in Abbildung 34 dargestellt ist. Zu jedem WebGazer-Datenpunkt gibt es eine gemittelte ($hardXMean, hardYMean$)-Koordinate und ein Label als *Fixation*, *Saccade* oder *Unclassified*.

4.4 Zwischenergebnis

Das Ergebnis dieses Kapitels ist ein klassifizierter WebGazer-Datensatz, der aus 16 Sessions besteht. Die beschriebene Methodik funktioniert. Sind die Daten erst einmal aufgenommen, lassen sie sich voll automatisch den einzelnen Abschnitten nach teilen und synchronisieren. Das erlaubt es, ohne Aufwand weitere Probanden hinzuzunehmen oder den Inhalt des Experiments auszuweiten und zu ändern.

Die Hardwaredaten weisen mehr Informationen auf, als in diesem Kapitel verwendet wurden. Die Ausschöpfung aller Informationen steht also noch aus. In diesem Kapitel wurde lediglich die Synchronisation auf Plausibilität geprüft. Sowohl das Auffinden der Kalibrierungsdaten (vgl. Abb. 31) als auch die Vergleiche der einzelnen Seiten überzeugen, dass die WebGazer- und Hardwaredaten richtig miteinander verbunden wurden.

Eine erste Analyse der Fixationen und Sakkaden ist allerdings ernüchternd. Die Verteilung der insgesamt 17955 Datenpunkte sieht wie folgt aus:

	Anzahl	Anzahl von Gruppen mit min. 3 Datenpunkten	Anzahl von Gruppen mit min. 5 Datenpunkten
Fixation	13458	8934	6097
Sakkade	3126	216	49
Unclassified	1371	255	80
Gesamt	17955	9405	6226

Es gibt deutlich weniger Sakkaden als Fixationen. Das ist zu erwarten. Eine Sakkade dauert durchschnittlich 30-80 ms und eine Fixation 200-300 ms (Holmqvist 2011, S. 23). In der Zeit einer Sakkade erhalten wir von WebGazer also ungefähr einen Datenpunkt, während es bei Fixationen 5 bis 6 Datenpunkte sind. Es kommt hinzu, dass wir ungefähr das Achtfache an Daten von der Hardware haben. Die Label werden nach Häufigkeit vergeben (vgl. Kapitel 3). Wenn also bei kürzeren Sakkaden ein WebGazer-Datenpunkt gesammelt wurde, gibt es in den Hardwaredaten trotzdem mit hoher Wahrscheinlichkeit viele Fixationspunkte. Das macht sich bemerkbar, wenn nach Gruppen von Fixationen, Sakkaden oder Unclassified gesucht wird. Gruppen von Sakkaden mit mindestens drei zusammenhängenden Datenpunkten

³⁷Hier richtet sich der Autor nach der Python-Bibliothek *Pandas*. Vgl. Dokumentation <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.idxmax.html>, abgerufen am 08.12.2020.

	timestamp	x	y	hardXMean	hardYMean	labelMax
0	125382	805.724813	598.797895	941.000000	521.250000	Fixation
1	125442	830.150531	575.433364	961.000000	553.333333	Fixation
2	125527	853.116613	571.833358	963.571429	553.714286	Fixation
3	125578	840.831675	574.131049	961.333333	558.500000	Fixation
4	125628	841.620069	571.197856	884.857143	409.857143	Saccade
5	125699	842.518537	575.987936	808.875000	253.500000	Fixation
6	125760	874.386016	589.937787	735.428571	243.000000	Saccade
7	125810	904.827436	570.694648	564.571429	208.142857	Fixation
8	125881	927.766794	559.070929	561.571429	207.714286	Fixation
9	125932	889.724340	488.029719	564.142857	217.000000	Fixation
10	125992	866.871291	433.887626	628.571429	221.714286	Saccade
11	126046	816.148375	365.926099	706.000000	221.000000	Fixation
12	126125	657.384092	300.498369	745.375000	221.125000	Saccade
13	126174	606.804005	263.939244	860.571429	242.000000	Fixation
14	126243	595.516829	220.890731	866.000000	252.400000	Unclassified
15	126310	572.425903	178.137413	NaN	NaN	Unclassified
16	126375	621.887242	183.424635	1043.750000	272.250000	Unclassified
17	126425	692.855484	279.142854	1027.375000	224.625000	Fixation
18	126509	714.817994	343.982016	1037.250000	210.000000	Saccade
19	126559	891.753827	514.969470	1163.857143	210.714286	Fixation
20	126626	1102.626432	683.825170	1189.875000	228.875000	Fixation
21	126697	1114.790076	765.276184	1178.875000	221.000000	Fixation
22	126759	1051.235349	734.680276	1117.000000	241.428571	Fixation
23	126809	1009.618173	686.000857	1113.142857	248.714286	Fixation
24	126880	1001.627717	650.583208	1241.625000	243.750000	Saccade

Abbildung 34: So sieht ein Auszug des Ergebnisses der Studie aus. Zu jeder (x , y)-Koordinate von WebGazer gibt es eine gemittelte Hardwarekoordinate ($hardXMean$, $hardYMean$) und dasjenige Label, welches in der jeweiligen Gruppe am Häufigsten vorkam ($labelMax$).

gibt es kaum und die Zahl verringert sich weiter, wenn die Gruppengröße auf fünf erhöht wird. Wirklich lange Sakkaden, wie noch in Kapitel 3.2 gibt es so gut wie gar nicht mehr. Das könnte den Schluss zulassen, dass WebGazer-Daten für das Finden von Sakkaden ungeeignet sind. Möglicherweise hängt das auch mit der sehr geringen Samplingrate der WebGazer-Daten in dieser Studie zusammen.

Es war auch zu erwarten, dass es bei den Fixationen nicht mehr so große, zusammenhängende und um einen Punkt konzentrierte Gruppen geben würde wie in Kap 3.2, weil die Probanden in der Studie einen größeren Freiheitsgrad in ihren Handlungen hatten. Allerdings zeigt die Abbildung 35, dass die Klassifikation der WebGazer-Daten als Fixationen ebenfalls schwierig wäre. Fixationsgruppen sind immer wieder von Sakkaden unterbrochen, die nur aus ein oder zwei Datenpunkten bestehen. So kommt es selten zu einer Gruppe mit vielen Datenpunkten. Noch seltener bilden diese Gruppen eine Fixation, die sich um einen Punkt konzentriert (vgl. Abb. 35, (1-4)), wie es von Fixationen zu erwarten ist. Am Häufigsten sind Fixationsgruppen zu erkennen, die aus zwei bis acht Punkten bestehen und wie eine Sakkade aussehen (vgl. Abb. 35), (5 - 12)). Diese können auch mehr Punkte auf einer kleineren Raumfläche enthalten (13, 14) oder sich über den ganzen Bildschirm ausbreiten (15, 16, 17).

Insgesamt ist es selten, dass eine Gruppe von Punkten (unabhängig vom Label) dicht beisammen liegt. Wenn das passiert, bestehen die Daten immer aus Fixationen, Sakkaden und unklassifizierten Punkten (vgl. Abb. 35, (18-20)).

Welche Rolle die Daten spielen, die vom Tobii-Pro-System als Unclassified klassifiziert wurden, ist noch unklar. Wie bei den Sakkaden tauchen sie häufig nur als einzelne Punkte auf oder bilden Gruppen von zwei bis vier Punkten, die eine Fixation oder Sakkade andeuten (vgl. Abb. 35, (21, 22)). Sehr selten bilden sie größere Gruppen, die in der Regel dicht beisammen liegen (23) und in noch selteneren Fällen über den Bildschirm verteilt sind (24).

5 Diskussion

Diese Bachelorarbeit hat sich mit der Klassifikation von Webcam-basierten Eyetracking-Daten in Fixationen und Sakkaden beschäftigt. Im ersten Teil wurde gezeigt, dass sich diese Unterscheidung in Daten, die mithilfe der JavaScript-Bibliothek WebGazer aufgenommen wurden, visuell und algorithmisch wiederfinden lässt. Die Methode der Datenaufnahme erlaubte die Daten mithilfe einer Heuristik zu klassifizieren, indem sie die Interaktion des Nutzers auf bestimmte Aktionen einschränkt. Im zweiten Teil der Arbeit wurden diese Beschränkungen aufgehoben. Dazu wurde eine Methode entwickelt, welche die Klassifikation der Daten nicht über eine Heuristik sondern mithilfe parallel aufgenommener Hardwaredaten ermöglicht. So ist ein Datensatz mit 16 Teilnehmern entstanden, der für jeden WebGazer-Datenpunkt eine gemittelte Hardware-Koordinate und ein Label als *Fixation*, *Saccade* oder *Unclassified* hat. Das Zusammenführen der Hardware- und WebGazer-Daten hat funktioniert. Bei dem Aufwand für die Vorüberlegungen und Vorbereitungen wäre es allerdings sinnvoll gewesen, eine längere Studie mit weiteren Stimuli durchzuführen. Es wäre ohne viel Aufwand möglich gewesen, weitere Grafiken einzufügen oder mehr Fragen zur Grafik zu stellen. Auch weitere Aufgaben wären denkbar gewesen. Das hätte mehr Daten generiert. Die meisten Probanden der Studie waren über ihre Kürze überrascht. Sie hatten zumindest mehr Fragen zu der Grafik erwartet. Da immerhin der gesamte Synchronisierungsschritt automatisch funktioniert, wäre es kein Problem mithilfe dieser Methodik neue, längere Studien durchzuführen. Es wäre sinn-

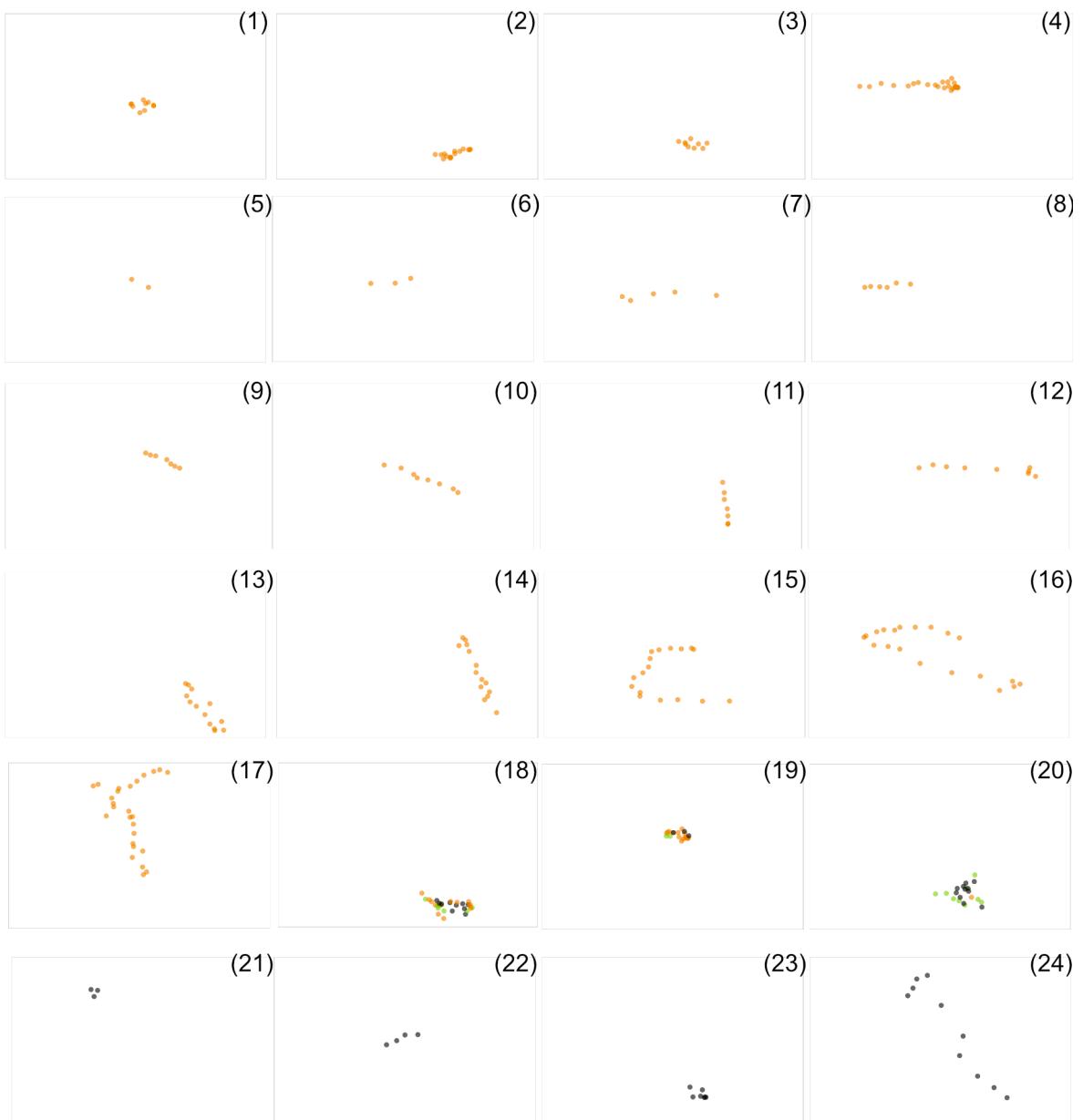


Abbildung 35: Auszüge aus den Daten vom zweiten Probanden der Studie. Dargestellt werden Fixationen (orange), Sakkaden (grün) und Unclassified (schwarz). (1-17) zeigen ganz verschiedene Arten von als Fixation klassifizierte Datengruppen. (18 - 24) stellen weitere Vorkommnisse dar.)

voll eine Studie zu erstellen, die Bilder verwendet, für die es schon Eyetracking-Daten gibt (Winkler und Subramanian 2013). Dann könnte WebGazer auch auf semantischer Ebene mit bereits vorhandenen Datensätzen verglichen werden. Für eine erneute Datenaufnahme wäre es sicher hilfreich, einen Experten für das Tobii-Pro System zu haben. Denn die Einstellungsmöglichkeiten sind vielfältig und es ist zu vermuten, dass sich die Daten der Hardware optimieren lassen.

Der entstandene Datensatz erfüllt die Kriterien für die in der Einleitung beschriebene Aufgabe, einen *Random Forest*-Klassifizierer zu trainieren und zu testen. Allerdings ist noch wenig über die Güte des Datensatzes bekannt. Ein erster Blick zeigt allerdings schon, dass sich nicht so konkrete Fixationen und Sakkaden wie in Kapitel 3.2 finden lassen. Zudem gibt es ein drittes Label *Unclassified*. Weitere Analysen werden testen müssen, ob WebGazer-Daten sich für die Klassifikation in Fixationen und Sakkaden überhaupt eignen. Zwar konnte schon in Kapitel 3 gezeigt werden, dass das unter konstruierten Umständen möglich ist (so auch bei Semmelmann und Weigelt 2018), aber es ist unklar, in wie weit das auf andere Daten zu trifft. Ein Analyseansatz könnte sein, die Algorithmen aus Kapitel 3.4 auf die WebGazer-Daten der Studie anzuwenden und die Label dann mit denen der Hardware zu vergleichen. Möglicherweise ist auch das Verfahren, die Label nach der Häufigkeit zu vergeben, zu überdenken.

Um Aufschluss über die Qualität von WebGazer im Vergleich zum Tobii-Pro-System zu bekommen, könnte geprüft werden, wie stark die WebGazer- von den Hardwarekoordinaten abweichen.

Außerdem könnte geprüft werden, in wie weit sich Datensätze unterscheiden, die nicht im Labor sondern unter unkontrollierten Bedingungen entstehen (Germine u. a. 2012). Bei diesen WebGazer-Daten fehlt dann zwar das Label, weil eine Übertragung von (fehlenden) Hardwaredaten nicht möglich ist, aber es können zumindest über die Streuung, die Geschwindigkeiten und die Samplingrate der Daten vergleichende Analysen angestellt werden. Wenn sich externe Parameter wie Lichtverhältnisse, Bildschirmgröße, Eingabegerät, Webcam-Qualität und auch das Verhalten der Probanden nicht kontrollieren lassen, sind andere Verteilungen der Daten zu erwarten. Dieses Argument hat auch einen direkten Bezug zu der Frage, welche Qualität die Klassifikation eines *Random Forest*-Klassifizierers hätte, der nur mit Daten aus einer Laborumgebung trainiert würde.

In dieser Arbeit war nicht genügen Raum, um den Datensatz aus Kapitel 4 hinsichtlich seiner Metadaten zu untersuchen. Dabei wäre vor allem die Frage nach der Qualität der Daten in Abhängigkeit von der Augenfarbe, der Hautfarbe oder dem Tragen einer Brille interessant. Weil diese Eigenschaften das Gesicht markant verändern, könnten Sie Einfluss auf die Gesichtserkennung und den daraus gewonnenen Daten haben.

Eine ganz andere Fragestellung könnte die semantische Analyse der Daten nachgehen. In dieser Arbeit wurde bewusst darauf verzichtet, zu prüfen, ob aus den WebGazer-Daten erkennbar ist, dass beispielsweise die Aufgabenbeschreibung gelesen oder wie die Grafik vom Probanden angeschaut wurde. Ließen sich solche (semantischen) Informationen aus dem Datensatz nicht ziehen, dann wäre die Klassifikation in Fixationen und Sakkaden hinfällig, weil sich daraus inhaltlich keine weiteren Zusammenhänge ergäben. Zu erwarten ist das allerdings nicht. Zwei Studien haben den sinnvollen Einsatz von WebGazer belegt (Papoutsaki, Laskey und Huang 2017), (Semmelmann und Weigelt 2018), wobei das Nachteile gegenüber Eyetracking-Hardware nicht ausschließt.

Die Hardwaredaten des Tobii-Pro-Systems liefern noch andere Daten, die in dieser Arbeit nicht weiter beachtet wurden. Sie erlauben aber möglicherweise aus den Daten weitere Features zu

extrahieren, die den WebGazer-Daten dann ebenfalls zugeordnet werden können. Diese können dann bei einem Ansatz mit maschinellem Lernen als weitere Parameter genutzt werden.

6 Literaturverzeichnis

Literatur

- Andersson, Richard u. a. (Mai 2016). "One algorithm to rule them all? An evaluation and discussion of ten eye movement event-detection algorithms". In: *Behavior Research Methods* 49.2, S. 616–637. DOI: 10.3758/s13428-016-0738-9.
- Discombe, Russell M und Stewart T Cotterill (2015). "Eye tracking in sport: A guide for new and aspiring researchers". In:
- Germine, Laura u. a. (Okt. 2012). "Is the Web as good as the lab? Comparable performance from Web and lab in cognitive/perceptual experiments." In: *Psychonomic bulletin & review* 19 (5), S. 847–857. ISSN: 1531-5320. DOI: 10.3758/s13423-012-0296-9. ppublish.
- Harezlak, Katarzyna und Paweł Kasprowski (2018). "Application of eye tracking in medicine: A survey, research issues and challenges". In: *Computerized Medical Imaging and Graphics* 65, S. 176–190.
- Holmqvist, Kenneth (2011). *Eye tracking: a comprehensive guide to methods and measures*. Oxford University Press. ISBN: 0199697086.
- Huang, Jeff, Ryen White und Georg Buscher (2012). "User see, user point". In: *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*. ACM Press. DOI: 10.1145/2207676.2208591.
- King, Andy J. u. a. (Jan. 2019). "Improving Visual Behavior Research in Communication Science: An Overview, Review, and Reporting Recommendations for Using Eye-Tracking Methods". In: *Communication Methods and Measures* 13.3, S. 149–177. DOI: 10.1080/19312458.2018.1558194.
- Papoutsaki, Alexandra (2018). "Democratizing Eye Tracking". Diss. Brown University.
- Papoutsaki, Alexandra, Aaron Gokaslan u. a. (Juni 2018). "The eye of the typer. a benchmark and analysis of gaze behavior during typing". In: *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*. ACM. DOI: 10.1145/3204493.3204552.
- Papoutsaki, Alexandra, James Laskey und Jeff Huang (März 2017). "SearchGazer". In: *Proceedings of the 2017 Conference on Conference Human Information Interaction and Retrieval*. ACM. DOI: 10.1145/3020165.3020170.
- Papoutsaki, Alexandra, Patsorn Sangkloy u. a. (2016). "WebGazer: Scalable Webcam Eye Tracking Using User Interactions". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. Hrsg. von Subbarao Kambhampati. IJCAI/AAAI Press, S. 3839–3845. URL: <http://www.ijcai.org/Abstract/16/540>.
- Piketty, Thomas (2014). *Das Kapital im 21. Jahrhundert*. DOI: 10.17104/9783406671326.
- Raimondas, Zemblys, Niehorster Diederick C und Holmqvist Kenneth (2018). *Using Machine Learning To Detect Events In Eye-Tracking Data. A Replication*. en. DOI: 10.5281/ZENODO.1343920.

- Salvucci, Dario D. und Joseph H. Goldberg (2000). *Identifying fixations and saccades in eye-tracking protocols*. DOI: 10.1145/355017.355028.
- Semmelmann, Kilian und Sarah Weigelt (2018). “Online webcam-based eye tracking in cognitive science: A first look”. In: 50, S. 451–465. ISSN: 1554-3528. DOI: 10.3758/s13428-017-0913-7.
- Seubert, Sabine (2019). *Visuelle Informationen beim Simultandolmetschen: Eine Eyetracking-Studie*. Bd. 47. Frank & Timme GmbH.
- Sülfow, Michael (2020). *Der Einfluss des Blickverhaltens auf die Urteilsbildung über Politiker*. Bd. 8. Nomos Verlag.
- Winkler, Stefan und Ramanathan Subramanian (2013). *Overview of Eye tracking Datasets*. DOI: 10.1109/qomex.2013.6603239.
- Xu, Pingmei u. a. (Apr. 2015). “TurkerGaze: Crowdsourcing Saliency with Webcam based Eye Tracking”. In: arXiv: 1504.06755 [cs.CV].
- Zemblys, Raimondas u. a. (2018). “Using machine learning to detect events in eye-tracking data”. In: *Behavior Research Methods* 50.1, S. 160–181. DOI: 10.3758/s13428-017-0860-3. URL: <https://doi.org/10.3758/s13428-017-0860-3>.

7 Anhang

Internetressourcen

- Datensatz aus Kapitel 3.2
<http://dschr.de/api/collectWebgazerData>
- Visualisierung des Datensatzes aus Kapitel 3.2
<http://dschr.de/infovis-project/>
- Quellcode zum Programm aus Kapitel 3.3
<https://gitlab.informatik.uni-halle.de/addvy/infovis-project>
- Kalibrierungstest von WebGazer
<http://dschr.de/CalibrationTest.html>
- Programm zur Datenaufnahme nach Kapitel 3
<https://collect-ml-data.herokuapp.com/>
- Quellcode zur Datenaufnahme nach Kapitel 3
<https://github.com/Adsata/collectMLData>
- Programm zur Datenaufnahme nach Kapitel 4
<https://collect-webgazer-data.herokuapp.com/>
- Quellcode zur Datenaufnahme nach Kapitel 4
<https://github.com/Daniel-Schreiber/webgazerStudy>
- Quellcode für Algorithmen aus 3.4
<https://github.com/Daniel-Schreiber/webcam-eyetracking-analyse>

CD

1. Bachelorarbeit
2. Ordner: Studie
Enthält den Studienleitfaden und das Studienformular sowie die Hardware- und WebGazer-Daten der Studie.
3. Ordner: Ergebnis
Enthält einen Ordner, indem für jeden Probanden die verknüpften Datensätze aus WebGazer und Tobii-Pro (siehe Kapitel 4) liegen. Außerdem gibt es eine Datei mit dem Namen *ResultCombineData.json*, die alle Datensätze vereint. Zusätzlich liegt im Ordner der Quellcode für die Methode der Verknüpfung der beiden Datensätze (*CombineData.py*) sowie zwei Jupyter-Notebooks, die über die Entwicklung Auskunft geben.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als angegeben verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Ort/Datum Unterschrift