



FACULTY OF TECHNICAL SCIENCES
Institute of Information Technology

Kevin Andreas Chromik

Videonavigation mit Wischgesteninteraktion

MAGISTERARBEIT

zur Erlangung des akademischen Grades
Magister der Sozial- und Wirtschaftswissenschaften

Studium
INFORMATIONSMANAGEMENT

Alpen-Adria-Universität Klagenfurt
Fakultät für Technische Wissenschaften

Begutachter:
Assoc.-Prof. Dipl.-Ing. Dr. Klaus Schöffmann
Institut für Informationstechnologie

Februar 2016

Eidesstattliche Erklärung

Ich versichere an Eides statt, dass ich

- diese eingereichte wissenschaftliche Arbeit selbstständig verfasst und andere als die angegebenen Hilfsmittel nicht benutzt habe,
- die während des Arbeitsvorganges von dritter Seite erfahrene Unterstützung, einschließlich signifikanter Betreuungshinweise, vollständig offengelegt habe,
- die Inhalte, die ich aus Werken Dritter oder eigenen Werken wortwörtlich oder sinngemäß übernommen habe, in geeigneter Form gekennzeichnet und den Ursprung der Information durch möglichst exakte Quellenangaben (z.B. in Fußnoten) ersichtlich gemacht habe,
- die Arbeit weder im Inland noch im Ausland einer Prüfungsbehörde vorgelegt habe und
- zur Plagiatskontrolle eine digitale Version der Arbeit eingereicht habe, die mit der gedruckten Version übereinstimmt.

Ich bin mir bewusst, dass eine tatsächwidrige Erklärung rechtliche Folgen haben wird.

Klagenfurt, im Februar 2016

Danksagung

An dieser Stelle möchte ich mich bei meiner Familie und insbesondere bei meiner Freundin für die intensive Unterstützung und Motivation während meines Studiums und allen Phasen dieser Arbeit bedanken. Ein weiterer Dank geht an Herrn Dipl.-Ing. Dr. Bonifaz Kaufmann für die Idee der Applikation. Ebenfalls gebührt ein besonderer Dank Herrn Assoc.-Prof. Dipl.-Ing. Dr. Klaus Schöffmann für die außerordentlich umfangreiche Betreuung dieser Arbeit. Abschließend möchte ich mich bei dem Institut für informationstechnologie Klagenfurt für die Unterstützung bedanken.

Kurzfassung

Durch die immer weitere Verbreitung von mobilen Geräten wie Tablets und Smartphones, haben sich auch die Ansprüche an Videoplayer und deren Navigationsmethoden verändert. In vielen Fällen ist die Standardnavigationsleiste, wie sie von den vorinstallierten Videoplayer-Applikationen eingesetzt wird, nicht befriedigend. Ansätze aus dem Bereich *automatischer Inhaltsanalyse* und *alternativer Inhaltsvisualisierung* können den Nutzern auf Desktop-Computern und mobilen Geräten bei Videobrowsing und Navigation einen erheblichen Vorteil verschaffen. Im Laufe der Arbeit werden verschiedene Grundlagen dieser Thematik aufgearbeitet und einige Applikationen, die Techniken und Methoden aus diesem Forschungsgebiet umsetzen, vorgestellt. Dies bildet das Fundament für die Anforderungen an den wischgestenbasierenden Videoplayer, der im Rahmen dieser Magisterarbeit implementiert wurde. Der vorgeschlagene Videoplayer, der mit Hilfe einer horizontalen Flick-Geste bedient wird, soll den Nutzer bei der Navigation innerhalb eines Videos unterstützen, ohne dabei vom Inhalt des Videos abzulenken. Dadurch soll erreicht werden, dass gesuchte Stellen schneller gefunden werden und weniger vom Inhalt übersehen wird. Im Rahmen einer Benutzerstudie wird der Prototyp mit einem Standardvideoplayer verglichen und evaluiert. Mit verschiedenen Suchaufgaben werden quantitative und mit einem Fragebogen qualitative Daten gesammelt. Mit den Ergebnissen aus deren Analyse sollen Aussagen über die Leistungsfähigkeit und die Nutzerakzeptanz getroffen werden.

Abstract

english Zusammenfassung

Inhaltsverzeichnis

Kurzfassung	v
Abstract	vi
Inhaltsverzeichnis	vii
Abbildungsverzeichnis	x
1 Einleitung	1
1.1 Ziel der Arbeit	1
1.2 Aufbau der Arbeit	2
2 Motivation und Grundlagen	4
2.1 Multimediale Inhalte	6
2.2 Videokonsum auf mobilen Geräten	8
2.3 Durchsuchbarkeit von Multimedia Dateien	9
2.4 Grundlagen der iOS Entwicklung	10
2.4.1 Model-View-Controller Konzept	11
2.4.2 Protokolle und Delegates	12
2.4.3 Observerpattern	13
2.4.4 AVFoundation	15
2.4.5 Gesturerecognizer und UIResponder	17
3 Videobrowsing	19
3.1 Navigationsunterstützung	19

3.1.1	Klassische Navigation	19
3.1.2	Automatische Inhaltsanalyse	21
3.1.3	Alternative Inhaltsvisualisierung und Navigation	25
3.2	Videobrowser Evaluierung	28
3.2.1	Video Browser Showdown	29
3.2.2	TREC Video Retrieval Evaluation	29
4	Videonavigation auf Tablets und Smartphones	31
4.1	Interface Design Guidelines	32
4.2	Multi-Touch Bildschirme und Gesten	32
4.3	Standardvideoplayer	35
4.4	Videobrowsing auf mobilen Geräten	36
5	Wischgestenbasierender Videoplayer	44
5.1	Spezifikationen	44
5.1.1	Anforderungsanalyse	44
5.1.2	Navigationskonzept	45
5.1.3	Grafische Benutzeroberfläche	47
5.2	Details zur Implementierung und Probleme	49
5.2.1	Hardware und Software	49
5.2.2	Probleme	49
5.2.3	Storyboard und Klassen	51
5.2.4	Umsetzung des Interaktionskonzepts	55
5.2.5	Logging	60
6	Benutzerstudie und Evaluierung	63
6.1	Aufbau der Studie	63

6.2 Durchführung	65
6.3 Evaluierung	67
6.3.1 Zielsuche	67
6.3.2 Zählaufgaben	70
6.3.3 Benutzerbewertung im NASA TLX	72
6.3.4 Bevorzugter Videoplayer	74
7 Resümee und Ausblick	76
Literatur	78
Appendix	86

Abbildungsverzeichnis

2.1	Opfer des Smartphone-Booms [1]	5
2.2	Weltweiter Tablet Absatz [2]	6
2.3	Prognostizierte Marktentwicklung von Tablets bis 2019 [3]	7
2.4	Objektorientierte Darstellung von Medientypen [4]	8
2.5	Yovisto mit Tagging Box [5]	10
2.6	Model-View-Controller [6]	11
2.7	Key-Value Observer anlegen [7]	14
2.8	Key-Value Observer anmelden [7]	15
2.9	Einordnung von AVFoundation in iOS [8]	16
2.10	Abspielen einer Videodatei mit AVFoundation [9]	16
2.11	UIResponder Chain [10]	18
3.1	Standardvideoplayer YouTube und VLC Player im Vergleich	20
3.2	Video Explorer mit verschiedenen Suchleisten [11]	23
3.3	Zoom innerhalb der Navigationsleiste beim Video Explorer [11]	24
3.4	Joke-O-Mat Benutzeroberfläche [12]	25
3.5	Hierarchische Baumübersicht [13]	26
3.6	YooSee Benutzeroberfläche [14]	28
4.1	Standardgesten für iOS [15]	34
4.2	VLC Player auf einem iPad	35
4.3	ThumbBrowser in Benutzung [16]	37
4.4	PocketDRAGON mit Fußballszene [17]	38

4.5	Keyframebasierter Navigationsbaum [18]	40
4.6	Mobile ZoomSlider Schnelldurchlauf durch Speedborders [19]	40
4.7	Wipe'n'Watch Player mit inter-Video Navigation [20]	42
4.8	SwiPlayer intra-Video Navigation und Granularität [21]	43
5.1	Navigation innerhalb von Textdokumenten [22]	46
5.2	Flickplayer Wischgesteninteraktion	47
5.3	Benutzeroberfläche des Flickplayers	48
5.4	Frame-Typen der Videokodierung [23]	50
5.5	Storyboard des Flickplayers	51
5.6	Geschwindigkeitsindikator Flickplayer	54
5.7	Player View Stack	56
5.8	Flickplayer Geschwindigkeits-Intervalle	58
5.9	Formular für Nutzertestinformationen	61
6.1	Prototypen mit Adaptionen für Studie	64
6.2	Aufbau der Benutzerstudie	65
6.3	Boxplot zur Suchdauer Nachrichten Videos	68
6.4	Boxplot zur Suchdauer Dokumentation Videos	69
6.5	Boxplot zur Suchdauer Quizshow Videos	70
6.6	Boxplot zur Suchdauer Fußball Videos	72
6.7	NASA TLX Bewertungsverteilung von 1 - 21	73
6.8	Subjektive Bewertung der Videoplayer	75

Seit der Einführung von Smartphones und Tablets im Jahr 2007 bzw. 2010, wurden diese zu einem wichtigen Bestandteil des Konsumentenmarktes. Viele verschiedene Produkte wie MP3-Player, Einsteigerkameras und Netbooks wurden vom Markt verdrängt, da deren Aufgaben von mobilen Geräten mit Bildschirmen, die eine Mehrfingererkennung ermöglichen, besser umgesetzt werden [24]. Diese Art von Verbraucherprodukten ermöglichen und verlangen völlig neue Methoden und Ansätze der Darstellung und Navigation von multimedialen Inhalten. Insbesondere bei großen Fotobibliotheken und langen Videos wurde mittlerweile erkannt, dass diese Stärken besser ausgenutzt werden können. Die Standardvideoplayer, das ist die standardmäßig installierte Software für die Video-betrachtung, setzen in der Regel auf eine zeitbasierte Leiste, auch bekannt als *Seekerbar*, um dem Nutzer die Möglichkeit zu bieten, sicher innerhalb eines Videos zu bewegen. Diese Navigationsleisten sind bei den meisten Videoplayern am oberen oder unteren Bildschirmrand positioniert und werden, zumindest auf mobilen Geräten mit berührungssempfindlichen Bildschirmen, mit den Fingern durch das Verschieben des Knopfes bedient. In vielen Fällen ist diese Art der Navigation vollkommen zufriedenstellend, jedoch gibt es Probleme beim Auffinden von Szenen in längeren Videos, insbesondere, wenn man nicht weiß, an welcher Stelle sich diese befinden. Es gibt viele Ansätze diese Probleme zu lösen, für die auch schon Prototypen existieren. Im Verlauf dieser Arbeit werden einige davon vorgestellt, sodass ein Überblick über die Vielfalt geschaffen wird. [25]

1.1 Ziel der Arbeit

Das Ziel dieser Arbeit ist es die Probleme von Standardvideoplayern zu analysieren und eine alternative Navigationsmethode für das Abspielen von Videos vorzustellen. Dies umfasst ebenso die Entwicklung eines Prototyps, welcher es ermöglicht mit Hilfe von Flick-Gesten durch Videos zu navigieren. Im Speziellen soll durch diesen Prototyp untersucht werden, ob sich intuitive Flick-Gesten besser für die Navigation innerhalb eines

Videos eignen, als standardmäßige Suchleisten. Die Wahl für eine horizontale Wischgesten wurde deshalb als Navigationsmethode gewählt, da ein Video als eine Reihe von Ereignissen betrachtet wird, durch die der Nutzer sich bewegt. Dabei wurde das Prinzip einer Listennavigation übernommen, da man sich dort ebenso schnell einen Überblick über den Inhalt einer Liste verschaffen kann. Die Bewegung sollte horizontal erfolgen, da sich somit der zeitliche Fluss eines Videos besser übertragen lässt. Auf Basis einer Studie, die im Rahmen dieser Arbeit durchgeführt wird, sollen Erkenntnisse über die Benutzerfreundlichkeit und Alltagstauglichkeit gewonnen werden. Dies geschieht insbesondere durch den direkten Vergleich mit dem Standardvideoplayer von iOS. Des Weiteren wird die subjektive Wahrnehmung der Benutzer erhoben, was ebenso in die Ergebnisse mit einfließen wird. Auf der Grundlage der Daten werden dann Antworten auf die Tauglichkeit und Benutzerfreundlichkeit der alternativen Interaktion im Gegensatz zur herkömmlichen Suchleistennavigation gegeben.

1.2 Aufbau der Arbeit

Da sich diese Arbeit mit Videosuche und Videobrowsing beschäftigt, werden, um ein besseres Verständnis der Thematik zu schaffen, angrenzende Themen mit einbezogen. Um die Relevanz hervorzuheben, wird im zweiten Kapitel die dahinterstehende Motivation detailliert erläutert. Da Smartphones und Tablet-Computer heutzutage sehr weit verbreitet sind, wird hierbei auch auf die wirtschaftlichen Aspekte eingegangen. Im Anschluss daran wird eine Definition für multimediale Inhalte gegeben, sowie ein Überblick, wie diese Medien auf mobilen Geräten konsumiert werden. Um die Details der Implementierung des Prototyps besser zu verstehen, sind noch Konzepte der iOS Entwicklung, wie Designpattern und Frameworks für Gestenerkennung und Video, aufzuarbeiten.

Im dritten Kapitel wird dann das Thema Videobrowsing intensiv aufgearbeitet. Im ersten Teil dieses Kapitels werden die verschiedenen Navigationsmethoden genauer erläutert, welche im Einzelnen die klassischen Methoden mit manueller Suchleiste, die automatische Inhaltsanalyse, sowie die Umsetzung von Videoplayern mit alternativer Inhaltsvisualisierung und Navigation sind. Um diesen Forschungsbereich weiter zu fördern, gibt es Evaluierungswettbewerbe. Der *Video Browser Showdown*, sowie *TRECVID* sind die beiden wichtigsten Veranstaltungen und werden aus diesem Grund genauer beleuchtet.

Das vierte Kapitel beschreibt alle Einzelheiten über Videobrowsing und Videonavigation auf mobilen Geräten. Dabei werden Besonderheiten der Benutzeroberfläche, Techniken von Multi-Touch Bildschirmen und verschiedene Gesten und deren Anwendungsfälle geauer erläutert. Des Weiteren wird dann speziell auf die Probleme von Standardvideoplayern eingegangen und einige Konzepte von Applikationen vorgestellt, die Lösungen

für diese Probleme liefern.

Da im Rahmen dieser Arbeit ein Prototyp einer mobilen Videoplayer Applikation implementiert wurde, werden die einzelnen Schritte dieses Prozesses im fünften Kapitel detailliert beschrieben. Zu Beginn werden im ersten Schritt die Spezifikationen genauer behandelt. Dies umfasst die Anforderungen an die Applikation, also welches spezifische Problem mit der Applikation gelöst werden soll, sowie eine Beschreibung des zu entwickelnden Navigationskonzepts und der Benutzeroberfläche. Im nächsten Schritt werden die Details für den Programmierteil der Applikation erläutert. Dabei werden zu Beginn die Probleme, die einen großen Einfluss auf die Art der Umsetzung hatten, erklärt und die Lösungen dazu vorgestellt. Des Weiteren werden einzelne Klassen und Methoden des Videoplayers beschrieben und welche Aufgaben diese erledigen. Dies ist Teil der Gesamtarchitektur des Projekts, welches ebenfalls im Ganzen genauer dargestellt wird. Da für die anschließende Benutzerstudie neben der qualitativen Untersuchung auch quantitative Daten erhoben werden sollten, musste ein Loggingsystem implementiert werden. Die Details zu dieser Aufgabe werden im letzten Teil des Kapitels über die Implementierung gegeben.

Die Evaluierung des Flickplayers, das ist der Name des wischgestenbasierten Video-players, ist ebenso Teil dieser Arbeit. Der Aufbau der Studie, sowie die Durchführung und Evaluierung und Präsentation der Ergebnisse folgt im sechsten Kapitel.

Die gewonnenen Erkenntnisse der Thematik und die Schlüsse, die aus der Studie gezogen werden können, werden im Anschluss resümiert und ein Ausblick für zukünftige Forschungsarbeiten gegeben.

KAPITEL

2 Motivation und Grundlagen

In diesem Kapitel werden die Grundlagen aufgearbeitet, die für das weitere Verständnis der folgenden Kapitel nötig sind, sowie die Motivation für die Arbeit erläutert. Als Einstieg wird das Thema aus wirtschaftlicher Perspektive beleuchtet, da die ökonomischen Aspekte aufgrund des Marktvolumens eine außerordentliche Bedeutung haben. Des Weiteren wird genauer erläutert was multimediale Inhalte sind und welche Schwierigkeiten es bei der Darstellung und Suche gibt. Im Anschluss werden noch verschiedene Methoden aus dem Bereich der Darstellung vorgestellt. Da die Entwicklung von iOS Applikation bestimmte Besonderheiten aufweist und um ein besseres Verständnis der Programm-details zu schaffen, wird in einem zusätzlichen Unterkapitel ein kurzer Überblick über Grundlagen der iOS Entwicklung gegeben.

Im Jahr 2007 hat Apple mit der Vorstellung des iPhones den gesamten Markt der Smartphones revolutioniert und damit die Art und Weise wie die Menschen das Internet benutzen, und unterwegs Videos und Musik konsumieren, für immer verändert. Der nächste Meilenstein wurde mit der Einführung des ersten iPads im Jahr 2010 gelegt, welches mit einer Bildschirmgröße von 9,7 Zoll als Alternative zu herkömmlichen Netbooks, also Notebooks mit kleinem Bildschirm und langer Akkulaufzeit, erwerblich wurde. Einer Studie [1] zufolge sind seit 2007 die Verkaufszahlen für Geräte wie Digitalkameras, MP3-Player und Navigationssysteme kontinuierlich geschrumpft und die der Smartphones immer weiter angestiegen. Die Grafik 2.1 zeigt, wie die Smartphones eine immer größere Rolle spielen. Das liegt insbesondere daran, dass Smartphones in der Lage sind viele der Funktionen out-of-the-box zu erfüllen, oder mit Applikationen die Funktionalität nachträglich zu installieren [26]. Da der Videokonsum auf den bis 2010 sehr kleinen Smartphone Bildschirmen nur bedingt befriedigend war, wurde mit dem iPad versucht, dieses Problem zu lösen. In Abbildung 2.2 ist der weltweite Absatz quartalsweise seit Q2 2010 abgebildet. Es ist deutlich zu sehen, wie der Absatz bis einschließlich 2012 stark gewachsen ist und sich seither auf einem nur leicht sinkenden Niveau hält. Die stark er-

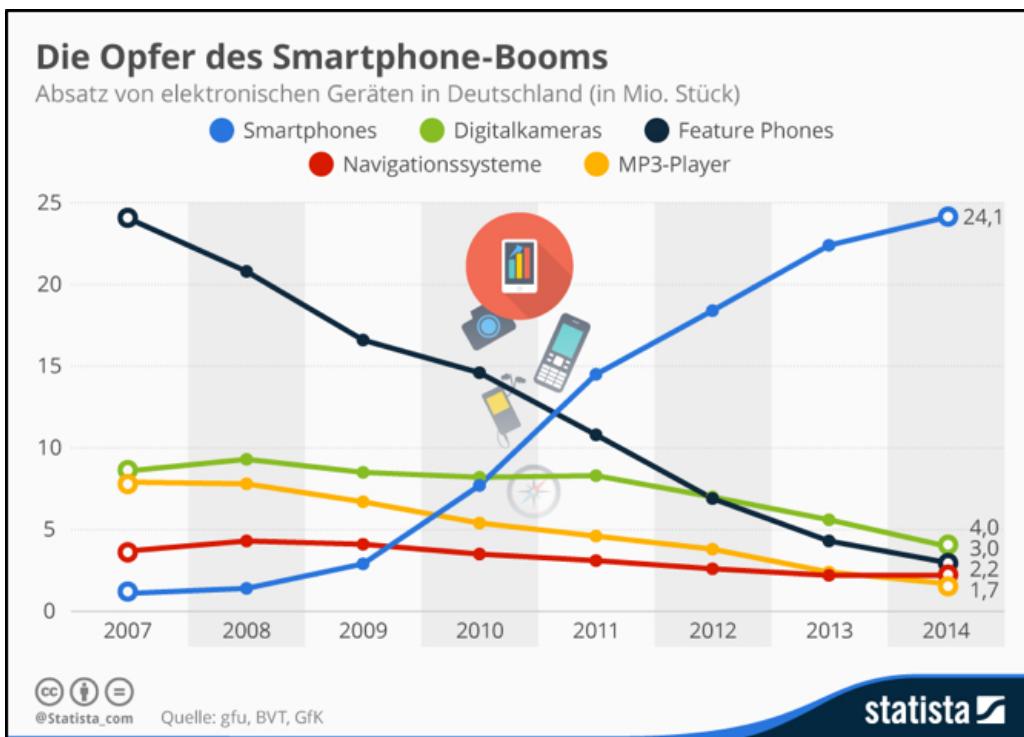


Abbildung 2.1: Opfer des Smartphone-Booms [1]

höhten Absätze jeweils im vierten Quartal sind insbesondere auf das Weihnachtsgeschäft zurückzuführen [2]. Apple ist mit dem iPad auf dem Tablet-Markt seit seiner Einführung Marktführer. Alleine im dritten Quartal 2015 wurden knapp 10 Millionen Geräte weltweit verkauft und erreicht somit einen Marktanteil von 20,3%. Auf dem zweiten Platz liegt der Koreanische Elektronikhersteller Samsung mit einem Marktanteil von 16,5%. Im Vergleich zum Vorjahr ist die Gesamtverkaufsmenge im dritten Quartal generell etwas geringer, die Marktanteile haben sich jedoch kaum verändert. [27] Analysten gehen jedoch davon aus, dass sich in den kommenden Jahren die Marktanteile weiter zugunsten von Microsoft mit ihren Windows-Tablets verschieben werden, da diese verglichen zu Android und iOS Geräten eher die Funktionen eines vollwertigen Laptops ersetzen können. Im Jahr 2015 liegt der Windows Marktanteil bei etwa 7% und soll laut Prognose im Jahr 2019 bei etwa 14,1% liegen. Generell geht die International Data Corporation davon aus, dass die weltweiten Verkaufszahlen sich bis 2019 konstant auf etwa 250 Millionen verkauften Tablets fixieren und sich das jährliche Marktwachstum einstellen wird. Diese Entwicklung ist auch in Abbildung 2.3 in einem Diagramm dargestellt. [3] Das iPad war in erster Linie für den Privatgebrauch gedacht, jedoch haben seither auch Un-

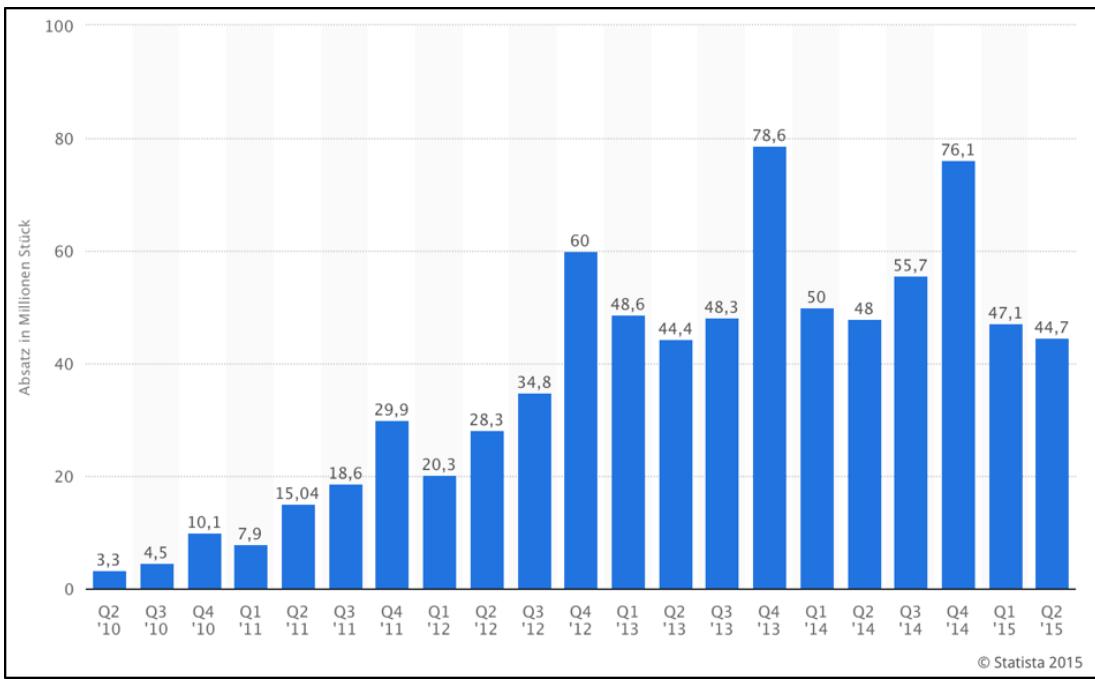


Abbildung 2.2: Weltweiter Tablet Absatz [2]

ternehmen großes Interesse an dieser Gerätetypologie gezeigt und evaluiert, wie diese Geräte den Alltag der Angestellten vereinfachen können, was insbesondere auf die hohe Mobilität zurückzuführen ist. Aus diesem Grund kommen die Nutzer sowohl aus dem Unternehmerbereich, sowie aus dem Privatsektor. In Schulen und Universitäten haben Tablet-PCs auch eine große Verbreitung gefunden. Technologiegestütztes Lernen kann die Leistung von Schülern und Studenten fördern, da die aktive Einbindung den Lernprozess unterstützt. Des Weiteren erhöht es die Interaktionsaktivität zwischen den Lehrern und Schülern. [28]

Die sehr hohe Konsumentenanzahl von Videomedien macht es deshalb interessant, die Art und Weise, wie das Medium konsumiert wird, zu analysieren und Vorschläge zu machen, wie diese Erfahrung verbessert werden kann.

2.1 Multimediale Inhalte

Der Begriff Medium bedeutet im Allgemeinen Informationsträger. So können Informationen, die in unterschiedlichster Form existieren, gespeichert, transportiert und verbreitet werden. Beispiele dafür sind Video, Audio, Text und Bilder [29]. Auf Tablets kann man viele unterschiedliche Arten von Medien konsumieren. Die hochauflösenden Bildschirme

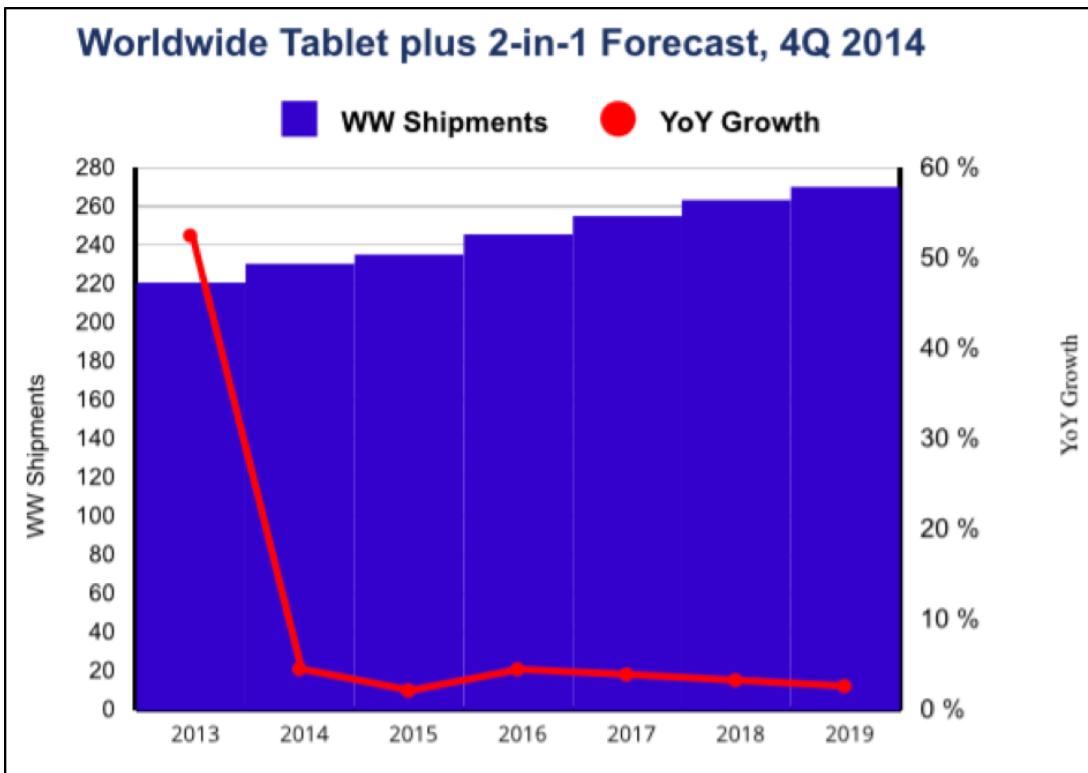


Abbildung 2.3: Prognostizierte Marktentwicklung von Tablets bis 2019 [3]

können Text so scharf darstellen, dass Bücher und Magazine selbst bei Dunkelheit angenehm gelesen werden können. Zusätzlich werden vollkommen neuartige Formen der Präsentation gegeben. Die Inhalte können nun auch mit Audio- und Videomaterial so kombiniert werden, sodass für den Nutzer ein tatsächlicher und unkomplizierter Mehrwert kreiert wird [30]. Doch nicht nur textueller Inhalt wird vermehrt auf mobilen Geräten mit berührungsempfindlichen Bildschirmen konsumiert, sondern auch Videos und Musik. Prinzipiell besteht multimedialer Inhalt aus zwei verschiedenen Komponenten. Zum einen ist das ein statischer Inhalt, wie beispielsweise ein Foto, was zu jedem gegebenen Zeitraum gleich ist. Zum anderen gibt es noch dynamische Inhalte, die sich zeitabhängig verändern, wie beispielsweise Animationen, audiomediale Inhalte und Videos. Darüber hinaus wird zwischen visuellen und nicht-visuellen Medien unterschieden. Visuelle Medien sind für den Betrachter sichtbar, wie Standbilder und Bewegtbilder. Dem gegenüber stehen die nicht-visuellen Medien wie Tonaufnahmen. Sobald mindestens zwei Medien miteinander kombiniert werden spricht man von Multimedia, wobei mindestens eines der Medien zeitbasiert, also kontinuierlich, sein muss. Ein klassisches Beispiel hierfür ist ein

Spielfilm, bei dem sowohl Audio und Video zu einem Stream zusammengeführt werden. In Grafik 2.4 ist veranschaulicht wie verschiedene Medienarten miteinander kombiniert

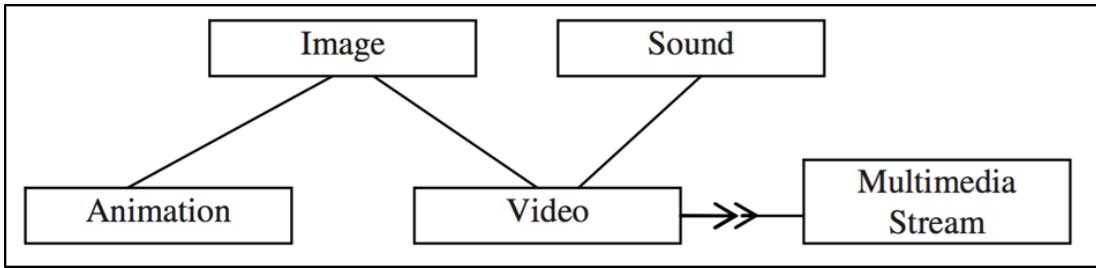


Abbildung 2.4: Objektorientierte Darstellung von Medientypen [4]

werden, zu einem Multimedia Stream. Die Abbildung gibt hier als Beispiel ein Video an, welches mit einer Audiospur unterlegt ist [4]. Standardmäßig sind Tablets mit Programmen ausgestattet, um solche Videodateien abspielen zu können. Welche Probleme im Kontext von Navigation und Darstellung existieren und mit welchen Methoden versucht wird diese zu lösen, wird in den folgenden Kapiteln aufgearbeitet.

2.2 Videokonsum auf mobilen Geräten

Der Konsum von Videos auf mobilen Geräten unterscheidet sich sehr von dem auf Desktop-PCs. Viele Fernsehsender bieten mittlerweile ihr Programm auch als Applikation an, sodass durch die oftmals permanente Internetverbindung auch unterwegs auf diese Medien zugegriffen und konsumiert werden kann. Medienquellen wie iTunes auf dem iPhone bzw. iPad oder auch Google Play für Android-Geräte sind große Konkurrenten zu den feststehenden Fernsehgeräten. Problematisch wird es, das zufriedenstellende Konsumerlebnis von den klassischen Videomedien, auf den mobilen Sektor zu übertragen, das liegt insbesondere daran, dass mobiler Videokonsum sich anders in den Alltag integriert [31]. Die Datenmengen, die über mobile Netzwerke konsumiert werden, sind heutzutage durch schnelle Internetanbindungen besonders hoch. Laut Cisco wird bis 2018 knapp 80% des weltweiten Datenverkehrs alleine durch Videostreaming erzeugt und bis dahin der mobile Datenverkehr in Bezug auf Volumen, den von kabelgebundenen Geräten überholt haben. Im Jahr 2013 lag dieser Wert zum Vergleich noch bei 66%. Plattformen wie YouTube und Netflix tragen hierbei eine ausschlaggebende Rolle, da Personen aufgrund von Bequemlichkeit diese Produkte eher auf Tablets und Smartphones konsumieren. Die Nutzer legen mittlerweile viel mehr Wert auf die Qualität der gelieferten Videos. Die Standardauflösung eines Fernsehers reicht daher oftmals nicht mehr aus, sondern Auflö-

sungen wie *FullHD*, also 1920x1080 Pixel, und sogar *UltraHD* was der 4-fachen Auflösung von FullHD entspricht, also 3840x2160 Pixel, werden heute von den meisten Streamingportalen unterstützt. Die hohen Displayauflösungen der mobilen Geräte haben diesen Bedarf aufkommen lassen. Daraus resultieren mehrere Herausforderungen, die es zu bewältigen gilt. Zum einen ist das die Bereitstellung der Videos auf zuverlässiger Weise, was bedeutet, dass insbesondere bei Videostreams das Video möglichst ohne Unterbrechungen bei Konsumenten ankommt und das bei einer gleichbleibend hohen Qualität. Zum anderen müssen diese Videos benutzerfreundlich wiedergeben werden, was dann von den Applikationen umgesetzt werden muss. Der Bereich, der sich mit dieser Thematik beschäftigt wird auch *Quality of Experience*, kurz QoE, genannt. [32]

Der Fokus dieser Arbeit liegt aber in der Durchsuchbarkeit und Darstellung der Inhalte, was in den folgenden Kapiteln weiter erörtert wird.

2.3 Durchsuchbarkeit von Multimedia Dateien

Große Videobibliotheken sowie lange Videos werden oft zum Problem, was die inhaltliche Suche betrifft. Wenn man nicht weiß wo sich eine Datei befindet oder wie diese inhaltlich strukturiert ist, kann dies den Nutzer frustrieren. Im Vergleich dazu können bei textuellen Inhalten wie digitale Bücher einfache Suchanfragen gestartet werden, ohne dass der Nutzer komplizierte Suchanfragen erstellen muss. Große Texte können daher entweder mit Hilfe eines vorhandenen Index durchsucht werden, wie einem Inhaltsverzeichnis, oder auch mit einer Volltextsuche. Bei Videos können solche Suchanfragen, auch Queries genannt, nicht formuliert werden, weshalb manuell durchsucht werden muss und dies unter Umständen äußerst zeitaufwendig sein kann. Dazu kommt noch, dass Nutzer aufgrund der enormen Datenmengen oft gar nicht wissen, wie sie das Objekt, das sie gerne finden würden, in einer Suchanfrage beschreiben sollen [33]. In der Regel werden die Videos mit Metadaten ergänzt, sowie Tags, also Schlüsselwörter, die den Inhalt des Videos in irgendeiner Weise logisch beschreiben. So lassen sich jedoch keine bestimmten Szenen textuell beschreiben, sodass der Videoplayer automatisch zur gesuchten Stelle springt. Kollaborativ werden solche Tags auch von mehreren Nutzern gesetzt, damit diese Annotation auch zeitbezogen geschieht. Diese Technik wird auch *Social Tagging* genannt. Ein Beispiel für ein solches System ist das Videoportal *Yovisto* [5], auf dem Videos aus akademischen Vorlesungen verfügbar sind. Der Videoplayer bietet verschiedene Möglichkeiten, um die Videos durchsuchbar zu machen. Neben dem klassischen Schieberegler, welcher die aktuelle Position der zu sehenden Szene in Relation zur Gesamtlänge zeigt, gibt es noch eine Tagging Box. Diese Tagging Box beinhaltet Schlagwörter, die die Nutzer passend zum Inhalt des Videos eingetragen haben. Da jeder Nutzer dieses Videoportals solche Annotationen machen kann, werden diese, im Falle von Wiederholungen entspre-

chend gewichtet. Daher sind Schlagworte mit häufigerem Vorkommen größer dargestellt, als Worte, die weniger oft genannt wurden. Die Abbildung 2.5 zeigt eine solche Tagging Box. Mit einem Klick auf eines der Schlagwörter öffnet sich darunter eine Liste mit

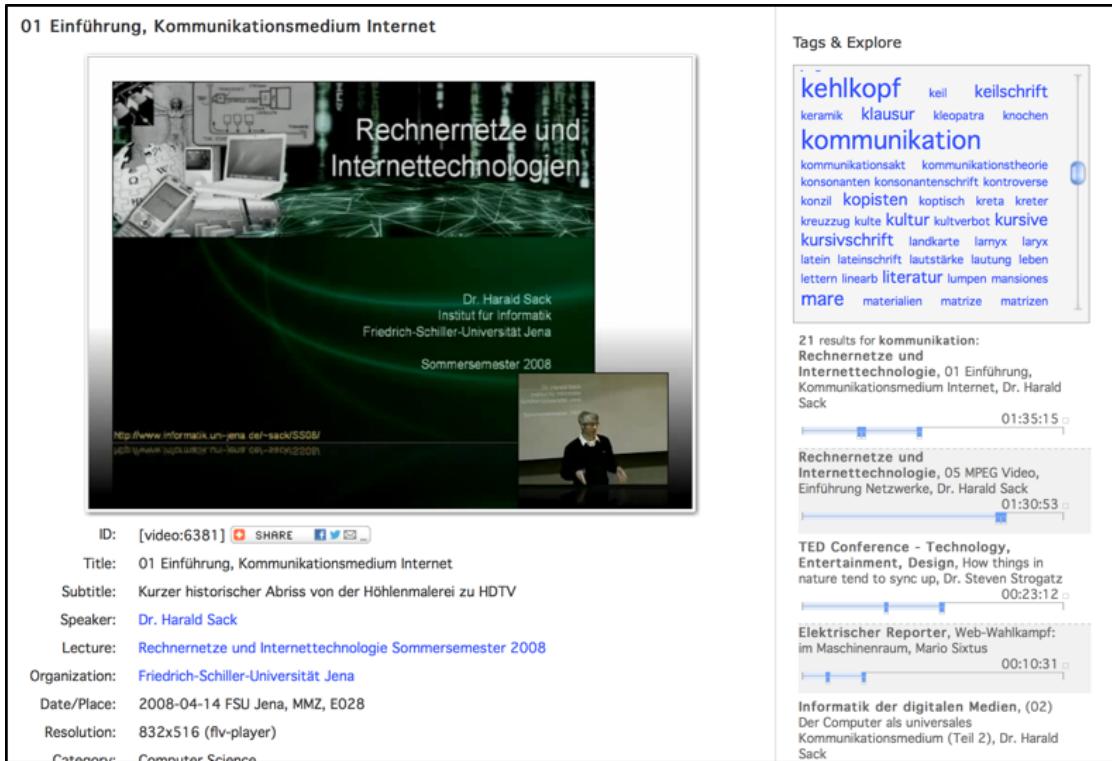


Abbildung 2.5: Yovisto mit Tagging Box [5]

Videos, in denen dieses Schlagwort annotiert wurde und jeweils an welchen Stellen im zeitlichen Verlauf der Videos. Von dort aus ist es dann möglich direkt zu der markierten Stelle zu springen, um sich den Inhalt anzusehen. Um das Thema oder den Inhalt des Videos schneller zu überblicken, werden unterhalb des Videos noch Metadaten wie der Name des Vortragenden, Aufnahmedatum, Untertitel, sowie der Name der Universität aufgelistet. [34]

2.4 Grundlagen der iOS Entwicklung

Die Entwicklung für iOS Geräte unterscheidet sich von Desktopanwendungen dahingehend, dass von Beginn die eingeschränkten Ressourcen mit einkalkuliert werden müssen. Um Applikationen für Apple's mobiles Betriebssystem entwickeln zu können, müssen

unterschiedliche Paradigmen verstanden werden. Die wichtigsten Konzepte, die für die Umsetzung des Flickplayers relevant sind, werden in diesem Kapitel behandelt.

2.4.1 Model-View-Controller Konzept

Eines der wichtigsten Paradigmen in der iOS Entwicklung ist das Model-View-Controllers Entwurfsmuster, da es tief in Cocoa Touch verwurzelt ist. Cocoa Touch umfasst das gesamte Framework, um iOS Applikationen zu entwickeln. Teile einer Applikation können in drei unterschiedlichen Kategorien eingeteilt werden, welche Model, View und Controller sind. Das Model beschreibt die Datenschicht, also alle Daten, die dem Programm zur Verfügung stehen und bearbeitet werden. In dieser Schicht ist man einzig und allein auf die Daten fokussiert, was bedeutet, dass man hier keine Informationen darüber speichert, wie diese Daten später dargestellt werden. Beispielsweise kann dies eine Datenbank sein, welche Teil der Applikation ist. Modelle werden in der Regel als Unterklassen von NS-Object erstellt, mit Instanzvariablen und Methoden, um deren Werte zu manipulieren. Die Visualisierung, also die Benutzeroberfläche, gehört zur sogenannten View. Auch hier

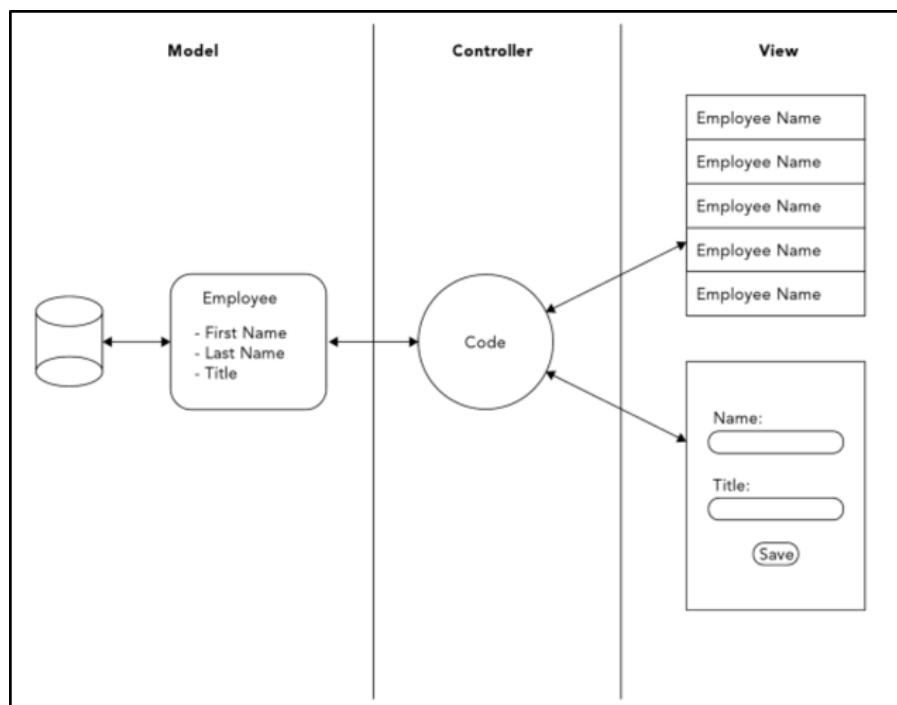


Abbildung 2.6: Model-View-Controller [6]

wird sich nur auf diese eine Aufgabe beschränkt und keine Logik implementiert, die zur

Abspeicherung der Daten benötigt wird. Die Kommunikation zwischen dem Model, also den Daten und der View wird vom Controller übernommen. Wenn sich beispielsweise die Daten ändern, dann wird auf dieser Ebene die Benutzeroberfläche darüber informiert, sodass diese sich dementsprechend anpassen muss. Wenn das Datenmodell besonders einfach ist, dann ist es dennoch möglich diese als Teil des Controllers zu implementieren, sodass keine zusätzliche Klasse dafür angelegt werden muss. In der iOS Entwicklung werden die Controller als Unterklasse von UIViewController angelegt. In Abbildung 2.6 ist das Konzept des Model-View-Controllers grafisch veranschaulicht. In diesem Beispiel gibt es eine Datenbank, welche eine Menge von Mitarbeitern verwaltet. Des Weiteren können die existierenden Mitarbeiter aufgelistet werden, sowie neue Mitarbeiter angelegt werden. Die strikte Modularisierung ist hier sehr gut zu sehen, da die View beispielsweise nur das Eingabeformular und einen Knopf zum Abspeichern anzeigt. Was mit den eingegeben Werte passiert, bzw. was generell passieren soll, wenn dieser Knopf gedrückt wird, wird allein vom Controller übernommen. Nach einer Aktion sind die neuen Daten ebenfalls nur in dem Model zu finden und nicht im Controller selbst. Der besondere Vorteil dabei ist, dass die Daten unabhängig verwaltet werden und so verschiedene Controller unterschiedliche Aufgaben übernehmen können. [6]

Da die Kommunikation zwischen der View und dem Model nie direkt passiert, müssen Benachrichtigungen mit sogenannten Protokollen und Delegates implementiert werden. Das folgende Kapitel gibt einen kurzen Überblick, welche Bedeutung diese in der iOS Entwicklung haben und wie sie umgesetzt werden.

2.4.2 Protokolle und Delegates

Protokolle und Delegates kommen dann zum Einsatz, wenn komplexe Kommunikation zwischen zwei Objekten gefordert ist. Die Verwendung von Delegates und Protokollen ist für die iOS Entwicklung essentiell und ist daher Teil der Grundlagen. Um dieses Konzept zu verstehen, lässt es sich am besten an einem Beispiel erklären. Eine TableView, also eine Liste mit mehreren Einträgen, durch die man navigieren kann, ist Teil eines Controllers. Die TableView soll eine Übersicht von Fotos haben und mit einem Tipp auf ein einzelnes Foto soll dieses vergrößert dargestellt werden. Die TableView besitzt eine Vielzahl von Methoden, die beispielsweise erkennen, auf welche Zelle getippt wurde oder auch ob der Nutzer durch die Liste navigiert. Da eine TableView aber nicht in jedem Fall in gleicher Weise darauf reagieren soll, wird die eigentliche Funktionalität von den referenzierten Objekten implementiert. Die Funktionen werden innerhalb der TableViews mit Protokollen definiert, die an den entsprechenden Stellen aufgerufen werden. Des Weiteren hat die TableView ein sogenanntes Delegate-Property, welches vom Typ des Protokolls ist. Auf diesem Delegate-Objekt werden dann innerhalb der TableView-Klasse

die Protokoll-Methoden aufgerufen. Wenn die TableView nun vom Controller initialisiert wird, dann wird der Controller zum Delegate dieser TableView-Instanz gesetzt. Dazu muss der Controller die Methoden, oder nur diejenigen, die auch tatsächlich benötigt werden, implementieren, um konform zum Protokoll zu sein. Wenn der Nutzer nun auf ein verkleinertes Foto in der Liste tippt, dann ruft die TableView die entsprechende Delegate-Methode auf und übergibt dieser die Position der Zelle. Da die Implementierung dieser Methode vom Delegate gemacht wird, in diesem Fall dem Controller, kann darin auf die Interaktion reagiert werden und mit der Information über die ausgewählte Zelle das entsprechende Foto vergrößern. Protokolle und Delegates werden nicht nur von Klassen aus dem iOS SDK eingesetzt, sondern können und sollten auch von eigenen Klassen umgesetzt werden. Der besondere Vorteil dieses Entwurfsmuster ist, dass sich so weitere Ableitungen der TableView-Klasse vermeiden lassen und sich so Beziehungen zwischen jeglicher Art von Klassen aufbauen lassen. Eine Kapselung der Logik wird so konsequent etabliert und die Logik nur dort implementiert, wo die konkreten Informationen auch verarbeitet werden. [35] [36]

2.4.3 Observerpattern

Neben den Protokollen und Delegates ist das Observer Entwurfsmuster ein ebenso wichtiger Bestandteil aus der Programmierung für iOS, um Benachrichtigungen bzw. Kommunikation zwischen mehreren Objekten umzusetzen. Der grundlegende Gedanke dahinter ist, dass an unterschiedlichen Stellen im Programm auf bestimmte Ereignisse reagiert werden muss. Wie auf diese Ereignisse reagiert wird, ist in diesem Fall zweitrangig. Prinzipiell haben zwei Arten von Observern einen besonderen Stellenwert in der iOS Entwicklung, welche zum einen die Notifications und zu anderen der Key-Value Observer sind. Die iOS Foundation bietet mit der NSNotificationCenter Klasse ein mächtiges Werkzeug, um mit solchen Benachrichtigungen umzugehen. Die grundlegende Idee dahinter ist, dass ein Objekt eine Notification hinzufügen kann, welche einen Namen hat. Das NSNotificationCenter bietet mit dem defaultCenter() eine Singleton Instanz, welche diese dann verwaltet. Um neue Benachrichtigungen hinzuzufügen, muss folgende Methode verwendet werden:

```
– (id<NSObject>)addObserverForName:( NSString *)name  
                           object:( id )obj  
                            queue:( NSOperationQueue * )queue  
                      usingBlock:( void ( ^ )(NSNotification *note ) )block
```

Andere Objekte können sich nun zu diesem Observer anmelden und eine Funktion implementieren, die ausgeführt wird, wenn zu dieser Notification ein Post ausgelöst wurde. Das Anmelden zu einer gegebenen Notification wird mit folgender Methode ausgeführt:

```
-(void)addObserver:(id)notificationObserver
    selector:(SEL)notificationSelector
    name:(NSString *)notificationName
    object:(id)notificationSender
```

Der Post, um alle Subscriber zu benachrichtigen, wird vom Objekt durchgeführt, welche die Notification erstellt hat. Folgende Methode muss dafür implementiert werden:

```
- (void)postNotificationName:(NSString *)notificationName
    object:(id)notificationSender
```

Man kann diese Art von Observern wie mit einem Newsletter vergleichen. Ein Dienst schickt einen Newsletter an alle Personen aus, die sich für diesen angemeldet haben. Der Dienst an sich macht hierbei keinen Unterschied darin, wie die einzelnen Personen darauf reagieren. Innerhalb von iOS werden viele dieser Notifications verwendet, wie beispielsweise von der Tastatur, die eine Notification postet, sobald diese zum Vorschein kommt oder wieder geschlossen wird. Auch andere Applikationsbenachrichtigung werden sehr häufig benutzt, sodass eine Applikation darauf reagieren kann, wenn sie in den Hintergrund geschoben wird. Bei Videoplayern wird dies in der Regel benutzt, um das aktuelle Video zu pausieren, sodass es nicht im Hintergrund weiterläuft. [37] [38]

Der Key-Value Observer hat eine besondere Bedeutung, da dieser unter anderem verwendet wird, um Benachrichtigung zwischen dem Model und dem Controller auszutauschen. Der Unterschied zur Notification ist hierbei, dass ein bestimmtes Attribut von einem Objekt überwacht wird und eine Benachrichtigung postet, sobald sich der Wert dieses Attributs ändert. Des Weiteren gibt es keine zentrale Stelle, die die Benachrichtigungen verwaltet. Mit Hilfe des folgenden Beispiels wird dieses Entwurfsmuster weiter erklärt. Es gibt zwei Objekte, das BankObject und das PersonObject. Das BankObject besitzt ein Attribut accountBalance, welches den aktuellen Kontostand einer Person wiederspiegelt. Die Instanz des BankObject fügt nun einen Observer hinzu, siehe Abbil-

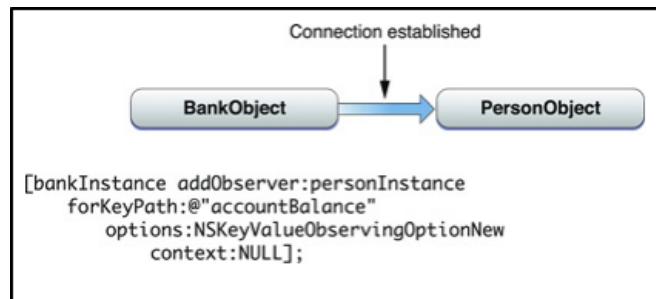


Abbildung 2.7: Key-Value Observer anlegen [7]

dung 2.7, welcher die Instanz des PersonObject benachrichtigt, sobald sich das Attribut accountBalance ändert. Wie die Instanz des PersonObject darauf reagiert, muss in der Klasse dieser definiert werden. Dazu muss die Methode in Abbildung 2.8 implementiert werden.

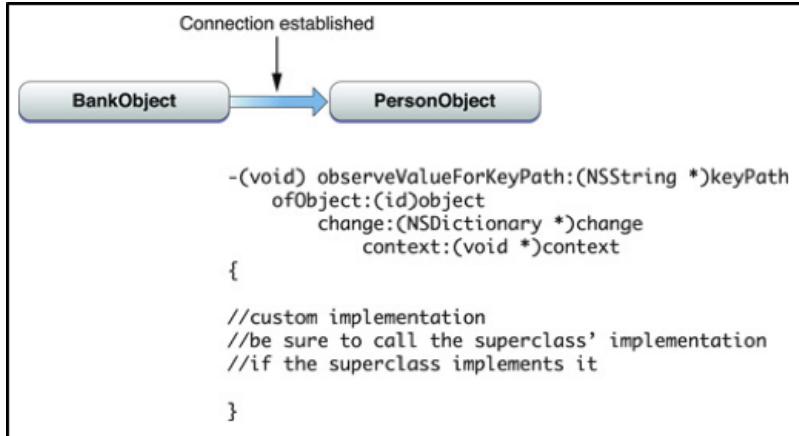


Abbildung 2.8: Key-Value Observer anmelden [7]

tiert werden. Der Methodenblock wird dann ausgeführt, sobald sich der Wert ändert. Der Vorteil hierbei ist, dass Benachrichtigungen nicht aktiv versendet werden müssen und Objekte ohne weitere Klassen miteinander verbunden werden und Informationen austauschen können. [7] [39]

2.4.4 AVFoundation

Für die Entwicklung des Flickplayers musste ein geeignetes Framework ausgewählt werden, welches in der Lage ist, Videos während der Laufzeit zu manipulieren. Das iOS SDK bietet mit der AVFoundation ein leistungsstarkes Framework, welches in der Lage ist, audiovisuelle Daten, also Ton- und Videoaufnahmen, zu erzeugen, editieren, untersuchen und selbstverständlich auch abzuspielen. Des Weiteren gibt es Klassen, die speziell den Umgang Audio implementieren. In Abbildung 2.9 ist zu sehen, wie die AVFoundation in iOS integriert wurde. Das Framework verwendet Methoden aus Core Audio, Core Media und Core Animation, welche zu den low-level Frameworks zählen [8]. Core Audio übernimmt hier beispielsweise alle Aufgaben, um die Audiodaten zu handhaben und Core Media unterstützt die AVFoundation mit dem Umgang von zeitbasierten Aktionen rund um CMTime [40]. Für das Abspielen eines Videos werden drei Komponenten benötigt. Zum einen ist dies ein AVAsset, also die eigentliche Datei, die abgespielt werden soll. Da dieses Objekt nicht direkt abgespielt werden kann, muss es einem AVPlayerItem übergeben werden, da dies den aktuellen Status des Videos repräsentiert. Der AVPlayer

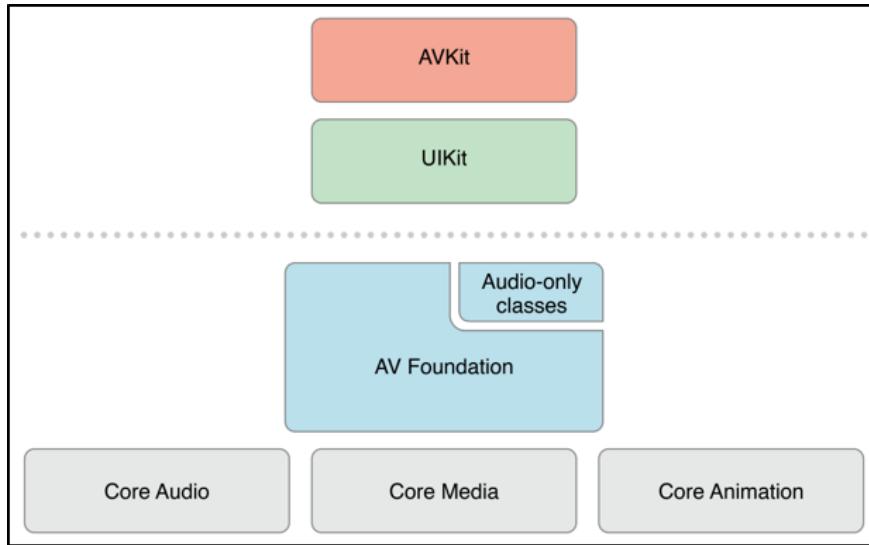


Abbildung 2.9: Einordnung von AVFoundation in iOS [8]

ist dann der eigentliche Videoplayer, welcher alle Methoden zur Verfügung stellt, um das Video zu steuern. Dies sind beispielsweise einfache Funktionen, wie das Starten und Pausieren des Videos. Um das Video für den Nutzer anzuzeigen, muss die AVPlayer-Instanz

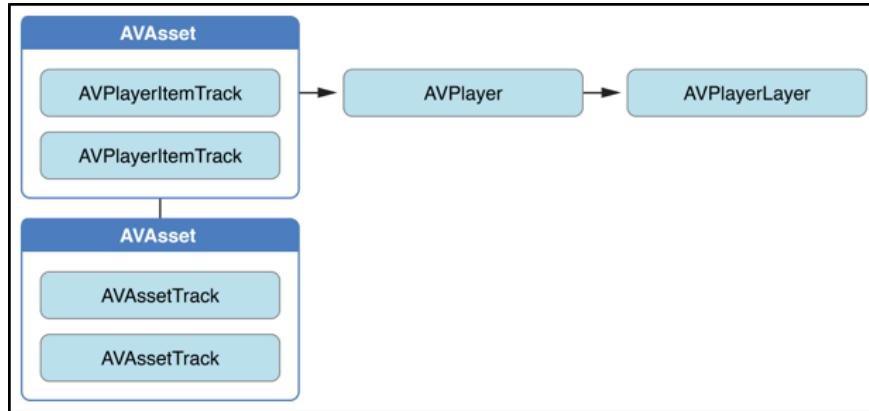


Abbildung 2.10: Abspielen einer Videodatei mit AVFoundation [9]

einem AVPlayerLayer übergeben werden, welcher dann in einer UIView angezeigt werden kann. In der Abbildung 2.10 sind die Referenzen zwischen den einzelnen Komponenten in einem Diagramm grafisch dargestellt. Da die eigentliche Datei, das AVAsset, nicht direkt abgespielt wird, sondern den Wrapper AVPlayerItem verwendet, kann eine Datei

von unterschiedlichen AVPlayern gleichzeitig abgespielt werden und somit das auch Rendering unterschiedlich sein kann. Das Manipulieren der Abspielgeschwindigkeit während der Laufzeit ist für den Flickplaner eine essentielle Funktionalität. Der AVPlayer hat hierfür ein Attribut *rate*, welches zu jedem Zeitpunkt verändert werden kann. Das geht sowohl zum Beschleunigen, zum Verlangsamten, sowie zum rückwärts Abspielen des Videos. All diese Funktionen sollten Teil des Flickplayers sein und somit auch angewendet werden. Dennoch gibt es hier Limitierungen, welche in den folgenden Kapiteln genauer beleuchtet werden. [9]

2.4.5 Gesturerecognizer und UIResponder

Fehlende Peripherie wie Maus und Tastatur, zumindest als Hardware, wird bei mobilen Geräten durch einen Berührungsempfindlichen Bildschirm ersetzt. Das iOS SDK bietet viele Schnittstellen, um auf Berührungen und Gesten reagieren zu können. Nicht jedes Objekt kann auf Eingaben reagieren, denn dazu muss es eine Unterklasse von UIResponder sein. Generell gibt es hier zwei unterschiedliche Ereignisse, welche zum einen ein Berührungsereignis und zum anderen ein Bewegungsereignis sind. Das Berührungsereignis gibt an, dass eine Berührung mit einem Objekt gerade begonnen oder aufgehört hat. Eine Berührung ist zu Ende, sobald der Nutzer den Finger wieder anhebt. Für die Implementierung der Flickplayers ist diese Art von Interaktionserkennung besonders wichtig, da aus diesen Informationen der Benutzereingabe die Interaktion erst möglich wird [41]. Die sogenannte Responder Chain, siehe Abbildung 2.11, also eine Kette, macht es möglich, Ereignisse durch eine Hierarchiekette weiterzugeben. Der Sinn dahinter ist, dass ein visuelles Objekt, in der Regel eine UIView oder eine Unterklasse davon, ein Event entgegennimmt und nicht selbst darauf reagiert. Das Event wird von der UIView in der Hierarchie weitergegeben und dann von dem Objekt verarbeitet, welches die Protokollmethoden implementiert [10]. Vordefinierte Arten von Gesten können mit einem sogenannten UIGestureRecognizer direkt einem UIView zugewiesen werden. Jede Unterklasse von UIGestureRecognizer befähigt den Umgang mit genau einer Gesteninteraktion. Die einzelnen Gesten, die vom iOS SDK zur Verfügung gestellt werden, sind im Folgenden aufgelistet.

- UITapGestureRecognizer
- UIPinchGestureRecognizer
- UIRotationGestureRecognizer
- UISwipeGestureRecognizer
- UIPanGestureRecognizer

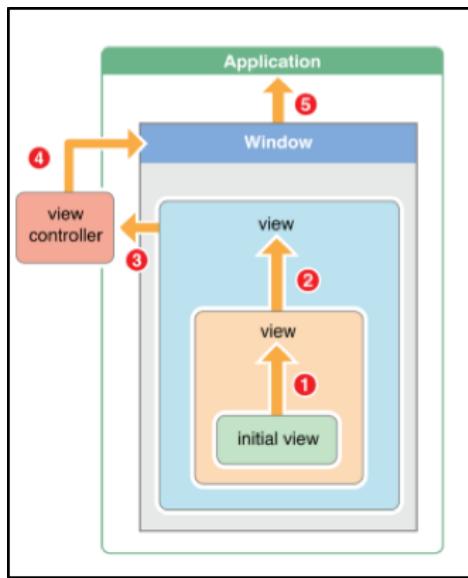


Abbildung 2.11: UIResponder Chain [10]

- UIScreenEdgePanGestureRecognizer
- UILongPressGestureRecognizer

Diese sind einfache Gesten, wie die Tap-Geste, oder auch auch die Pinch-Geste, die insbesondere verwendet wird, um ein Objekt zu vergrößern bzw. zu verkleinern. Zusätzlich kann hier noch angemerkt werden, dass eine Instanz mehrere Attribute besitzt, welche beispielsweise angeben, mit wie vielen Fingern die Geste ausgeführt werden muss, um als solche erkannt zu werden. Eine Instanz von UIView hat die Methode addGestureRecognizer, mit der man eine Instant der oben genannten GestureRecognizer hinzufügen kann. Des Weiteren benötigt die Methode einen Selector, welcher ausgeführt wird, wenn der Nutzer diese Geste auf dem Objekt gemacht hat. [42]

In diesem Kapitel wurden die Grundlagen aufgearbeitet, um die Implementierung des Flickplayers möglich zu machen. Wie der Prototyp im Detail umgesetzt wurde, wird in Kapitel 5 behandelt. Im folgenden Kapitel wird ganze Thematik von Videobrowsing detailliert erörtert.

Im vorherigen Kapitel hat sich herauskristallisiert, dass sich multimediale Dateien nicht ohne Weiteres durchsuchen lassen, jedenfalls nicht, wenn man eine vollständig manuelle Suche vermeiden möchte. Das Gebiet des Videobrowsings beschreibt die Aufgabe durch ein Video zu navigieren, um eine oder mehrere Szenen zu finden, oder einen Überblick über den Inhalt zu bekommen. Daher gibt es die Möglichkeiten den Inhalt zu analysieren und grafisch aufzubereiten, die Darstellung und Navigation an sich zu ändern, oder beide Methoden miteinander zu kombinieren [43]. In diesem Kapitel werden deshalb Methoden vorgestellt, die eine oder beide dieser Konzepte einsetzen.

3.1 Navigationsunterstützung

Im Bereich der Videoplayer wurden bisher einige Versuche gemacht, die Navigation zu vereinfachen, sodass auch Nutzer ohne besondere Vorkenntnisse das Werkzeug bedienen können. Insbesondere bei mobilen Geräten wie Smartphones und Tablets gibt es nur wenige Alternativen zur klassischen Bedienoberfläche. Die geringe Leistung und der Formfaktor, also die geringe Bildschirmgröße, spielen hier eine nicht unerhebliche Rolle [44].

3.1.1 Klassische Navigation

Mit der Einführung von Videorekorder, engl. *Video Cassette Recorder*, wurde auch der Begriff *VCR controls* etabliert. Diese beschreiben die Standardeingabemethoden, um einen dem Videorekorder bedienen zu können. Diese waren üblicherweise Knöpfe, welche Funktionen wie Start, Pause, vor- und zurückspulen und Stopp übernahmen. Dieses Konzept hat sich mittlerweile so stark bei den Nutzern eingeprägt, dass es von allen gängigen Video und Musikplayern übernommen wurde. Dies wurde teilweise auch mit einer Zeitleiste kombiniert, sodass auch grafisch sichtbar wird, wo man sich momentan im Video befindet [45]. Als Beispiele zeigen die Abbildungen in 3.1 die Benutzerober-

fläche zweier bekannter Videoplayer. Zum einen ist das der aktuelle *YouTube Player* und zum anderen der *VLC Media Player* in der Version 2.2.1 für MacOS. Sowohl der YouTube Player, als auch der VLC Media Player haben eine sehr ähnlich strukturierte Menüführung. Beide haben in der unteren linken Ecke den Start- bzw. Pause-Knopf, sowie die Möglichkeit zum nächsten Video in der Wiedergabeliste zu springen. Des Weiteren besitzen beide die übliche Navigationsleiste, um sich schnell innerhalb des Videos zu bewegen. In Ergänzung dazu hat der YouTube Player eine Funktion, bei der der

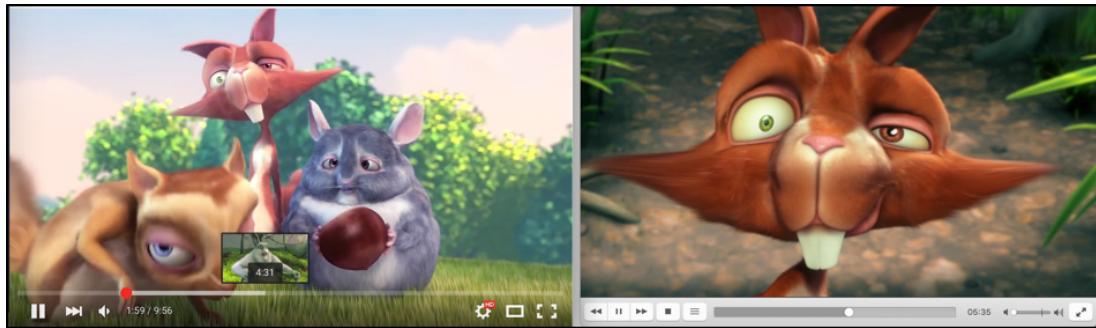


Abbildung 3.1: Standardvideoplayer YouTube und VLC Player im Vergleich

Nutzer mit dem Mauszeiger über die Suchleiste geht und ohne zu klicken eine Vorschau dieser Stelle zu bekommen. Wenn der Nutzer eine bestimmte Szene finden möchte, aber nicht weiß an welcher Stelle sich diese befindet, ist diese Navigationsmethode unter Umständen weniger zufriedenstellend. Anhand eines Beispiels lässt sich das Problem von zeitleistenbasierten Videoplayern besser hervorheben. Ein Fußballspiel wird live übertragen und während der Halbzeitpause kann man die vergangene Spielhälfte nochmals zur Analyse abspielen lassen. Für den Zuschauer sind hier offenbar nur wichtige Szenen interessant, die einen Einfluss auf das Spielergebnis haben. Das sind vor allem Tore, Fouls oder Angriffe, die fast zu einem Tor geführt haben. Bei einem 45-Minuten langem Video würde eine manuelle Suche sehr zeitaufwendig sein, da man sich in der Regel die exakten Zeitpunkte nicht merken konnte. Es gibt Videoplayer, die es während der Wiedergabe eines Videos oder Streams ermöglichen, Markierungen zu setzen. Innerhalb der Halbzeitpause kann dann von einer Markierung zur nächsten navigiert werden, ohne sich unwichtige Stellen ansehen zu müssen. Die Granularität der Zeitleisten ist oftmals eher ungenau und lässt den Nutzer in Sekundenintervallen navigieren. Das liegt daran, dass bei besonders langen Videos die Zeitleiste aufgrund von Platzmangel auf dem Bildschirm nicht entsprechend skaliert werden kann. Die maximale Bewegungsgenauigkeit auf dem Bildschirm liegt bei einem Pixel. Die Distanz von einem Pixel zum nächsten ist daher bei kurzen Videos ebenfalls kürzer, gemessen an dem relativen Verhältnis zur Videolänge.

Dies wird zum Problem, wenn man sich eine bestimmte Szene stark verlangsamt anschauen möchte, um eine detaillierte Analyse zu machen. Das ist zum Beispiel der Fall, wenn der Ball außerhalb des Spielfeldes gelangt und es nicht sicher ist, welcher Spieler den Ball zuletzt berührt hat. Auch hier müssen Videoplayer eingesetzt werden, die über die herkömmliche Funktionalität hinausgehen und beispielsweise eine Navigationsmethode implementieren, die eine Frame-by-Frame Navigation ermöglicht, die Zeitleiste skalierbar ist, oder das Video schlichtweg langsamer abspielen lässt. [25]

In den folgenden Kapiteln werden noch weitere Tools für den Desktop-Bereich vorgestellt, die diesen konventionellen Pfad verlassen und mit Hilfe von Inhaltsanalysen und interaktiver Videosuche diesen Vorgang beschleunigen und vereinfachen.

3.1.2 Automatische Inhaltsanalyse

Bei sehr großen Ansammlungen von Multimediadaten ist eine manuelle Durchsuchbarkeit praktisch nicht gegeben. Mit Hilfe einer automatischen Inhaltsanalyse kann der Nutzer ein Video oder mehrere Videos nach bestimmten Merkmalen untersuchen, je nach dem was in dieser Situation am sinnvollsten ist. Dazu kommt noch, dass dieses Teilgebiet von Videobrowsing in der Regel nicht nur von Nutzern verwendet wird, die schnell eine bestimmte Szene innerhalb eines Videos finden möchten, sondern auch von Firmen die beispielsweise Videoüberwachungssysteme einsetzen. Denn hier steht man vor dem Problem, dass teilweise rund um die Uhr gefilmt wird und es dadurch unmöglich ist, bestimmte Aktivitäten aus dem Video hervorzuheben, die für den Überwacher von Bedeutung sind. Zum Beispiel kann ein solches Video auf Bewegungsaktivitäten untersucht werden, sodass man einfach und schnell von einer aktiven Szene zur anderen navigieren kann, ohne sich den uninteressanten Inhalt dazwischen anschauen zu müssen. Da heutige Systeme zwar schon sehr ausgereift sind, kann und wird man dennoch nicht vollkommen auf menschliche Begutachtung verzichten können. Die für den Betrachter interessanten Szenen zu erkennen ist eine Sache, jedoch müssen die von dem Videobrowser gelieferten Ergebnisse auch selbst einfach und in angemessener Zeit durchsuchbar sein [46]. Daher ist eine grafisch sinnvolle Benutzeroberfläche des Tools ebenso ausschlaggebend wie der Algorithmus, der diese Ergebnisse produziert. Der ganze Vorgang wird auch Video Summarization genannt, da die wichtigen Inhalte aus dem langen Video zusammengefasst und aufbereitet werden [47] [48]. Im Folgenden werden verschiedene Tools vorgestellt:

Juengling et al. [43] haben den *VideoZoom* vorgeschlagen. Veränderungen werden hier in zwei Phasen erkannt. Zuerst werden zur Laufzeit Änderungen von angrenzenden Bildern hervorgehoben. Dies wird in der zweiten Phase mit einem Algorithmus kombiniert, welcher die Schatten von den Objekten erkennt. So ist es möglich die interessanten Objekte zu markieren, die dann auch Tubes genannt werden. Im Anschluss kann man

solche Tubes als Suchinformation übergeben und VideoZoom liefert eine Übersicht zurück, mit allen gefundenen Objekten. Mit einem Klick auf ein Ergebnis, welches ein Vorschaubild der Szene ist, navigiert man im originalen Video zur gesuchten Stelle. Im Fall der VideoZoom Studie lieferte insbesondere die Suche nach Fahrzeugen sehr genaue Ergebnisse.

Der *Video Explorer* von Schöffmann et al. [11] erlaubt es mit zwei Arten ein Video zu durchsuchen. Zum einen können gesuchte Features durch Histogramme visuell dargestellt werden und zum anderen kann das Video anhand von gegebenen Beispielen durchsucht werden. Bis dahin konnte man Videosuchtools in drei verschiedene Klassen aufteilen. Zum einen ist das die textuelle Suche, bei der Schrift und Sprache innerhalb der Videos extrahiert wird und als Text zur Verfügung steht, womit sich dann gewöhnliche Textsuchen durchführen lassen. Die zweite Kategorie ist, dass man ein Beispielbild als Suchparameter verwendet. Problematisch ist jedoch, dass der Nutzer oftmals kein Beispiel zur Hand hat, das er verwenden kann. Als dritte Kategorie gibt es noch die konzeptuelle Suche, welche es erlaubt semantische Konzepte als Suchparameter zu übergeben. Das bedeutet, dass der Nutzer beispielsweise nach Strand sucht und alle Stellen im Video angezeigt werden, die einen Strand zeigen. Dies geschieht anhand von Deskriptoren, die Farbe, Texturen und Kompositionen innerhalb des Videos mit einem trainierten Machine Learning Algorithmus zusammenfassen und damit das Video durchsuchen können. Ein großes Problem dabei ist, dass die Analyse eines gegebenen Videos um ein vielfaches länger dauert, als die eigentliche Videolänge. Dazu kommt noch, dass die Navigationsoberflächen solcher Tools kaum intuitiv sind, was eine Suche noch zeitaufwendiger macht. Um dieses Problem zu lösen, implementiert der Video Explorer eine Farb- und Bewegungserkennung, die während der Dekodierung des Videos schnell durchgeführt werden kann und projiziert die Ergebnisse als Histogramme zu einer Zeitleiste, die gleichzeitig als Navigationsleiste dient, wie sie auch bei Standardvideoplayern verwendet wird. In Abbildung 3.2 ist die Benutzeroberfläche des Video Explorers mit drei verschiedenen Features zu sehen, nach denen gesucht wird. Da wären zum einen die Key Frames, die dominante Farben und der Bewegungsverlauf innerhalb des Videos. Jedes einzelne Feature wird auf einer eigenen Suchleiste projiziert. Anhand des gelben Balkens weiß der Nutzer sofort, wo er sich im Video gerade befindet. Im gezeigten Beispiel ist eine Nachrichtensendung zu sehen. Hier werden verschiedene Berichte abgespielt, die immer zuvor vom Nachrichtensprecher angekündigt werden. Da sich in der Regel innerhalb der selben Sendung die Kleidung und der Hintergrund des Sprechers nicht verändern, kann mit Hilfe der aktivierte Suchen für die dominanten Farben immer sehr einfach auf der Navigationsleiste erkannt werden, wo ein Beitrag zu Ende ist und eine neuer Beitrag beginnt. Hier sind die dominanten Farben Orange und Schwarz sehr deutlich zu erkennen. Die Bewegungserkennung ist in diesem Fall auch sehr hilfreich, da sich der Nachrichtensprecher nicht

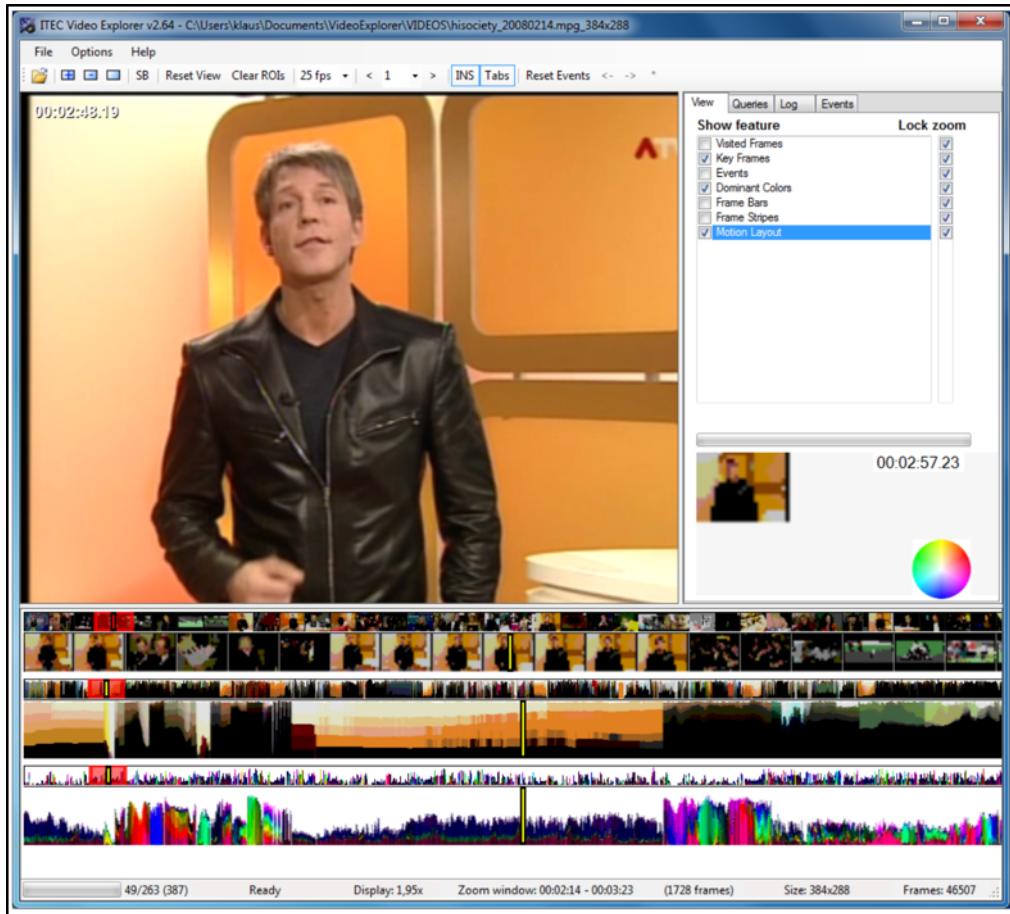


Abbildung 3.2: Video Explorer mit verschiedenen Suchleisten [11]

viel bewegt und auch keine großen Kamerenschwenker gemacht werden. Auf der Navigationsleiste für diese Suche werden Bewegungen ebenfalls durch Farben dargestellt, wobei Weiß bedeutet, dass keine Bewegung vorhanden ist. Eine weitere Eigenschaft des Video Explorers ist, dass man innerhalb der Navigationsleiste hineinzoomen kann, wie in Abbildung 3.3 zu sehen ist. Das ist besonders hilfreich, wenn das Video etwas länger ist und man eine detailliertere Übersicht einer ausgewählten Sequenz bekommen möchte. Wie schon angesprochen unterstützt der Video Explorer auch eine Beispielsuche. Dafür wird ein gewünschter Bereich des Videos markiert und später dann durch den Player angezeigt, an welchen Stellen im Video dies das gesuchte Muster wiederholt. [11]

Für Zielsuchaufgaben, bei denen der Nutzer vorher schon weiß, wie die gesuchte Stelle aussieht, hat Bai et al. [49] einen Videobrowser vorgeschlagen, der zu Beginn eine Videosammlung analysiert und Keyframes extrahiert. Die gewonnenen Informationen werden in

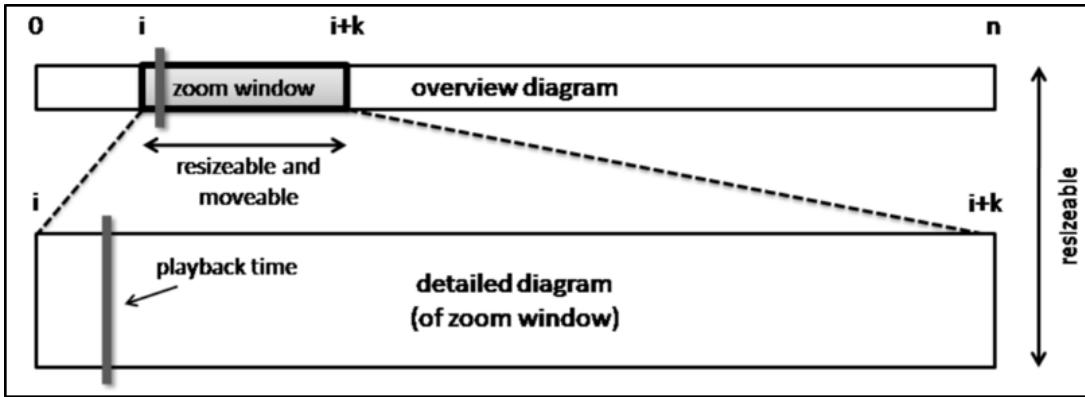


Abbildung 3.3: Zoom innerhalb der Navigationsleiste beim Video Explorer [11]

eine Datenbank gespeichert, die für die späteren Suchen herangezogen wird. Wenn ein Nutzer nun eine Szene finden möchte, dann wird die gesuchte Stelle als Suchparameter dem Suchsystem übergeben, welches passende Keyframes mit deren Zeitstempeln zurückliefert. Dieses Ergebnis wird dann vom Nutzer verwendet, um an die gesuchte Stelle zu navigieren.

Mit den *Joke-O-Mat* von Friedland et al. [12] wurde ein Ansatz vorgestellt, bei dem die Audiospur einer Sitcom-Serie analysiert wird und auf dieser Grundlage verschiedene Szenen erkannt werden können. Die Analysealgorithmen können trainiert werden, sodass bestimmte Personen erkannt werden können und man mit Hilfe der Zeitstempel dann zu den Szenen springen kann, an denen die gesuchten Personen gerade Sprechen. In Abbildung 3.4 ist die Benutzeroberfläche des Joke-O-Mat's dargestellt. Die Audioanalyse ist so entwickelt worden, dass auch das Lachen der Zuschauer in der Serie erkannt wird. Davon ausgegangen, dass nach einem Satz eines Darstellers, gefolgt von Gelächter, ein Witz gemacht wurde, konnten diese Szenen auch als solche markiert werden. Der Nutzer kann damit ebenfalls von einem Witz zum nächsten springen. Der Joke-P-Mat wurde speziell am Beispiel der Serie Seinfeld entwickelt, was auch aus der Benutzeroberfläche ersichtlich ist. In der rechten Spalte können alle Schauspieler selektiert werden, nach denen gesucht werden soll, sowie eine geschlechtsspezifische Suche. In der unteren Navigationsspalte gibt es verschiedene Menüs, die beispielsweise eine Übersicht über alle Witze gibt. Mit einem Klick auf einen Bildausschnitt gelangt der Nutzer direkt in diese Szene. Mit einer gewöhnlichen Navigationsleiste, sowie einem Knopf zum Abspielen, bzw. Pausieren ist man jederzeit in der Lage, manuell zu navigieren. Der Joke-O-Mat zeigt sehr gut, dass Videonavigation nicht nur auf Grundlage des visuellen Teils gemacht werden kann, sondern auch andere Komponenten eines Multimediums mit einbezogen

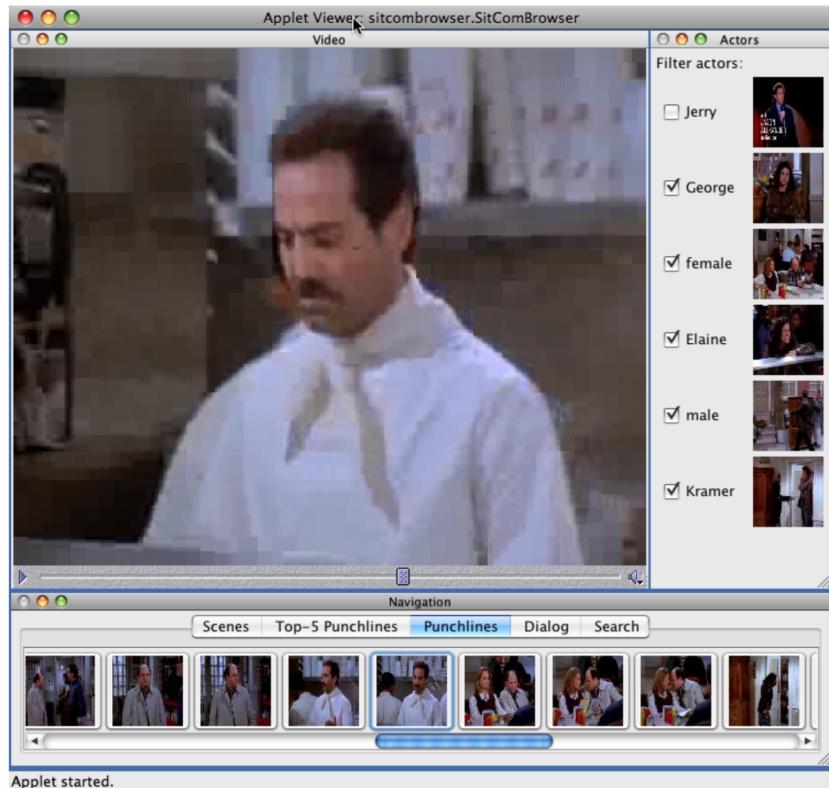


Abbildung 3.4: Joke-O-Mat Benutzeroberfläche [12]

werden können.

Im nächsten Kapitel werden Alternativen zur Inhaltsanalyse vorgestellt. Dies ist die Möglichkeit durch Veränderung der Visualisierung des Inhaltes und erweiterter Navigation ebenso die Durchsuchbarkeit zu vereinfachen und beschleunigen.

3.1.3 Alternative Inhaltsvisualisierung und Navigation

Bisher bieten nur wenige Videoplayer eine Alternative zur klassischen Navigationsleiste. Wie im Kapitel zur klassischen Navigation schon gezeigt wurde, hat der YouTube Player ein kleines Vorschaufenster, welches anzeigt, wo man im Video hinspringen wird. Neben der algorithmischen Analyse von Videos ist oftmals eine andersartige Darstellung des Inhalts und auch Navigation vollkommen ausreichend, um dem Nutzer einen Vorteil zu verschaffen. Vor allem wenn der Nutzer keine geeignete Suchanfrage machen kann, da er nicht weiß, wie man das zu Suchende beschreiben soll [50]. Ein besonderes Kriterium für Massentauglichkeit ist eine einfache Handhabung des Tools. Zu flache Lernkurven

können den Nutzer frustrieren und zum Aufgeben bewegen. Es gibt einige Versuche dieses Dilemma zu bewältigen, von denen im Folgenden einige vorgestellt werden. [51]

Der *Baumbasierte Video Browser* von del Fabro [13] erlaubt es innerhalb kürzester Zeit sich einen Überblick über den Inhalt eines Videos oder Videobibliothek zu verschaffen. Am Beispiel eines einzelnen Videos wird nach dem Öffnen eine Übersicht von Vorschaubildern angezeigt, die jeweils alle eine identisch lange Zeitspanne abbilden. Die Anzahl an Vorschaubildern ist frei konfigurierbar, sodass man beispielsweise eine 3 mal 3 Zellen Übersicht einstellen kann. Bei einem Video mit 90 Minuten Länge steht jedes der Bilder also für 10 Minuten des Videos. Mit einem Klick auf eines der Bilder gelangt man eine Stufe tiefer in der Hierarchie, wodurch nun wieder 9 Vorschaubilder angezeigt werden, die den Inhalt repräsentieren, welcher von dem ausgewählten Bild abgedeckt wurde. In dieser Hierarchieebene werden also diese 10 Minuten wieder aufgeteilt, sodass jedes Bild etwas über eine Minuten des Videoinhalts anzeigen. Abbildung 3.5 zeigt wie sich der Baum verzweigt und so der Inhalt des Videos sichtbar wird. Das besondere

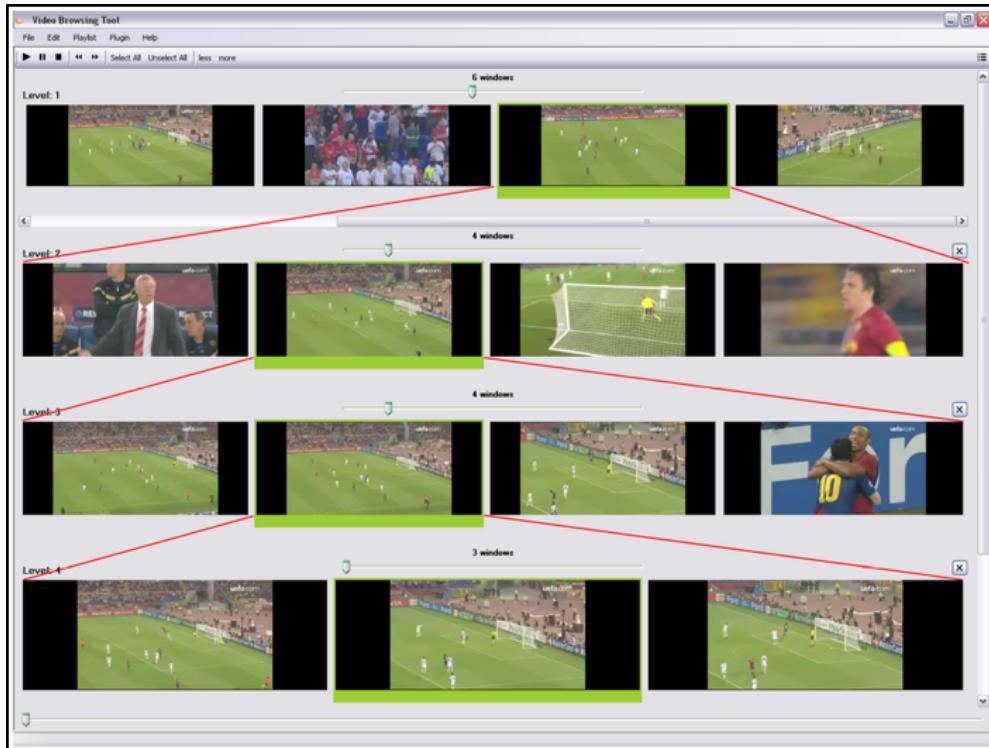


Abbildung 3.5: Hierarchische Baumübersicht [13]

ist noch, dass der Nutzer mit Hilfe von Schieberegeln durch alle sichtbaren Einzelsequenzen navigieren kann und so die zu suchende Stelle schnell auffindbar machen kann.

Die Audiospur ist nur in der Einzelübersicht aktiviert, da beim Abspielen von mehreren Tonspuren gleichzeitig ein schlechtes Benutzererlebnis entsteht. Der Videoplayer wurde so implementiert, dass eine Anbindung an ein Inhaltsanalysetool möglich ist.

Der *3D Video Browser* von Müller et al. [50] kombiniert eine 3-dimensionale Darstellung des Inhalts mit einer interaktiven und animierten Navigation. Ähnlich wie der baumbasierte Videobrowser von del Fabro [13] ist hier der Ansatz das Video oder die Videosammlung hierarchisch darzustellen. Durch die 3-dimensionale Anordnung lassen sich die verschiedenen Ebenen aber beliebig im Raum verteilen. Ansätze von 3D-Darstellungen gab es schon zuvor, jedoch unterstützten diese keine hierarchische Navigation, sondern waren allein auf die Darstellung der Navigationsleiste beschränkt [52]. Beim 3D Videobrowser hat der Nutzer zudem die Möglichkeit zwischen 8 verschiedenen Darstellungsoptionen zu wählen. Mit einem Klick auf ein Segment wird die Szene abgespielt. Innerhalb der Anordnung kann man sich zudem völlig frei im Raum bewegen und den gesamten Inhalt erkunden. Der 3D Videobrowser ist auch in der Lage eine komplette Videobibliothek in dieser Form abzubilden und navigierbar zu machen. [50]

Ein dritter Videobrowser, welcher den Inhalt an sich nicht verändert darstellt, sondern nur eine erweiterte Navigationsleiste implementiert, ist der *ZoomSlider* von Hürst und Barvers [53]. Der Grundgedanke ist, dass bei sehr langen Videos eine Navigation mit herkömmlicher Suchleiste oft unpräzise ist da beim Bewegen des Schiebereglers die Sprünge zwischen den einzelnen Sequenzen zu groß sind und der Inhalt dadurch nicht erfasst wird. Um dieses Problem zu beheben kann man während dem Anschauen des Videos das Mausrad drehen, um in die teilbasierte Navigationsleiste hineinzuzoomen. Dadurch wird der Bewegungsradius eingeschränkt und der Nutzer kann verfeinert den Inhalt erkunden und weniger Stellen werden übersprungen. Durch das nach unten drehen des Mausrades wird der Balken wieder hochskaliert, sodass die ursprüngliche Navigation wieder möglich ist. Die Benutzeroberfläche gibt während der Bedienung eine Rückmeldung und zeigt an, wie stark man in die Suchleiste hineingezoomt hat. Ein besonderer Vorteil dieser Navigationsmethode ist, dass große Videodateien genauer untersucht werden können, ohne den Vollbildmodus zu verlassen.

Azzopardi et al. [14] haben einen Videobrowser entwickelt, der sich aufgrund der Einfachheit in der Bedienung speziell an Kleinkinder im Alter zwischen zwei und sechs Jahren richtet. Die Benutzeroberfläche ist ähnlich wie ein Karussell aufgebaut. Abbildung 3.6 gibt eine Übersicht über den sogenannten *YooSee* Browser. Jede der drei sichtbaren Reihen entspricht einer Kategorie, die in einer Administrationsoberfläche zuvor festgelegt werden müssen. Diese sind vergleichbar mit einer Wiedergabeliste. Das mittlere Video ist dieses, welches gerade abgespielt wird. Mit einem Klick auf das Vorschaubild rechts bzw. links daneben, wird ein Video zum gleichen Kontext abgespielt. Um zu einer anderen Kategorie zu wechseln, kann das Kind auf die obere oder untere Reihe klicken. Da der

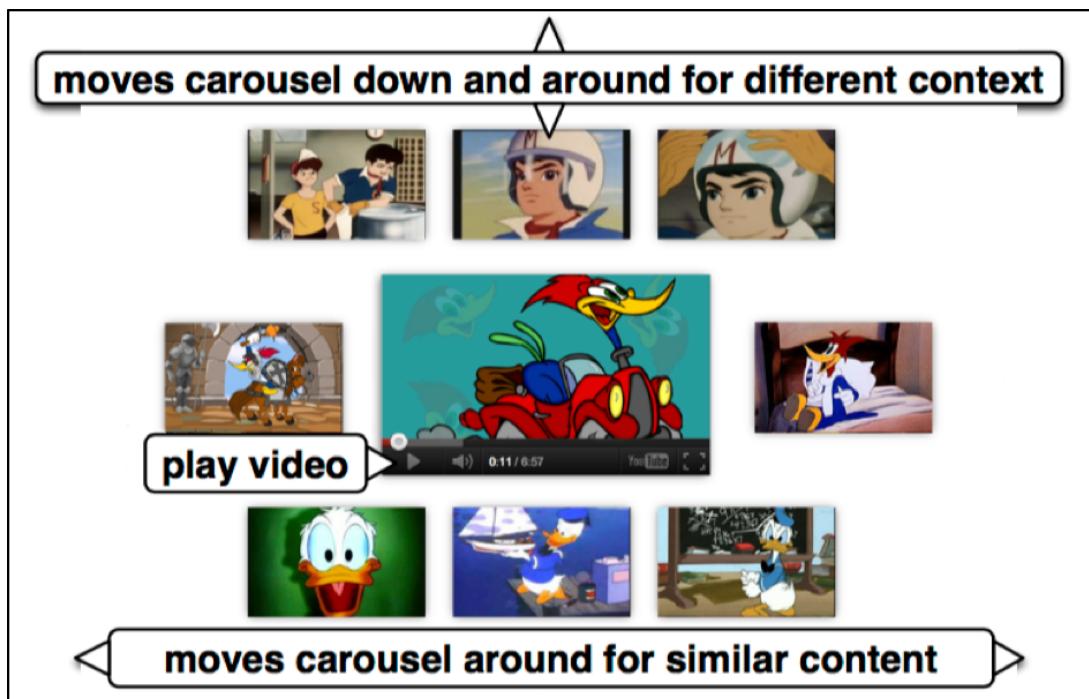


Abbildung 3.6: YooSee Benutzeroberfläche [14]

Videobrowser zum Abspielen Youtube Videos verwendet, wird auch der YouTube Player für eine Navigation innerhalb eines Videos benutzt.

Da es sehr aufwändig ist solche Videobrowser einheitlich zu evaluieren, wurden dafür verschiedene Wettbewerbe ins Leben gerufen. Das folgende Kapitel zeigt die zwei wichtigsten Wettbewerbe, die sich in dieser Thematik spezialisiert haben.

3.2 Videobrowser Evaluierung

Benutzerstudien eignen sich sehr gut, um neue Ideen in der Praxis zu untersuchen. Da jedoch verschiedene Stärken und Schwächen haben und die Testumgebung stark variieren kann, beispielsweise durch unterschiedliches Videomaterial, Auswahl der Testkandidaten, gibt es den Bedarf eine homogene Umgebung zu schaffen, um verschiedene Videobrowser miteinander zu vergleichen. Im Rahmen von Wettbewerben wird den Forschern die Möglichkeit geboten ihre Videobrowser in einer einheitlichen Umgebung gegeneinander antreten zu lassen und so die Leistungsfähigkeit vergleichbar zu messen. In diesem Kapitel werden die zwei bekanntesten dieser Art vorgestellt und zwar der *Video Browser Showdown* und *TREC Video Retrieval Evaluation*.

3.2.1 Video Browser Showdown

Um die Forschung im Bereich der interaktiven Videosuche bzw. *Interactive Video Retrieval* weiter voranzutreiben, wurde im Jahr 2012 der erste Video Browser Showdown als Sonderveranstaltung der International MultiMedia Modeling Conference ins Leben gerufen. Vor allem sollen Methoden und Ideen umgesetzt werden, die sich auf den Nutzer orientieren. Der Wettbewerb findet seither auf jährlicher Basis statt. Im Allgemeinen sehen die Aufgaben so aus, dass die Teilnehmer eine Reihe von Ausschnitten gezeigt bekommen, welche sie in einem längeren Video, oder auch in einer Videosammlung finden müssen. Da der Fokus des Wettbewerbs vor allem darin liegt, interaktive Suchverfahren voranzutreiben, sind automatische Suchaufgaben, wie beispielsweise eine reine Textsuche, nicht erlaubt. Vielmehr geht es darum, durch alternative Navigation und Filtern des Inhaltes, die gesuchte Stelle so schnell wie möglich zu finden. Da alle Teilnehmer gleichzeitig in den gleichen Videos suchen, lassen sich so die Stärken und Schwächen herauskristallisieren. Die gezeigten Ausschnitte haben eine Länge von 20 Sekunden, welche jeweils aus einem in etwa einstündigen Video extrahiert wurden. Dazu zählen die Aufgaben zur Kategorie KIS, also *Know-Item-Search*, denn die Teilnehmer wissen wie das zu suchende Objekt aussieht, wissen aber nicht wo es zu finden ist. Alle Tools sind an den VBS Server angeschlossen, welcher die Zeit misst, bis eine Lösung abgegeben wird. Dazu kommen noch sogenannte Penalties, also Strafzeiten, die bei dem Abgeben einer falschen Lösung verteilt werden. Die Zeiten werden auf Punkte pro Aufgabe umgeschichtet, woraus sich schließlich am Ende durch Addition eine Gesamtpunktzahl ergibt. Mit dieser wird dann der Gewinner ermittelt. Da die Entwickler der Videobrowser selbst Experten sind, was deren Bedienung angeht, werden insgesamt zwei Durchläufe gemacht. Die erste Runde ist die sogenannte Expertenrunde, bei denen die Entwickler selbst die Aufgaben durchführen. Die zweite Runde, die Anfänger Runde, bei der freiwillige Konferenzteilnehmer die Bedienung übernehmen, welche alle Experten im Bereich von Multimedia sind. In den ersten beiden Jahren war der VBS auf eine Stunde begrenzt, aber durch den hohen Bedarf mehr Daten und Erkenntnisse zu sammeln, wurde im Jahr 2014 die Dauer auf 5 Stunden ausgedehnt. Dadurch konnten viel mehr Aufgaben durchgeführt werden, denn pro Aufgabe wird ein 3-minütiges Zeitfenster gegeben. Viele verschiedene Systeme wurden mittlerweile seit 2012 präsentiert. Gewonnen haben sowohl Browser mit komplizierten Algorithmen, als auch Applikationen, die nur auf intuitive Bedienung gesetzt haben. [51] [54]

3.2.2 TREC Video Retrieval Evaluation

Die TREC Video Retrieval Evaluation, kurz *TRECVID*, wurde im Jahr 2001 von der Disruptive Technology Office (DTO) und dem National Institute of Standards and Tech-

nology (NIST) ins Leben gerufen. Des Weiteren erhält die Konferenz Unterstützung von verschiedenen Instituten der US Regierung. Die Ziele der TRECVID Konferenz sind die Forschung im Bereich der Informationsgewinnung auf Basis von besonders großen Video Sammlungen zu fördern, sowie die Kommunikation zwischen Wissenschaft, Industrie und der Regierung zu steigern. Dabei sollten Ideen ausgetauscht werden, um den Fortschritt weiter zu unterstützen. Durch den Austausch von Ideen sollen die Technologien eine weitere Verbreitung und Anklang in der Gesellschaft finden, welche letztendlich dabei helfen sollen, die Probleme aus der realen Welt zu lösen. Im Gegensatz zum Video Browser Showdown liegt hier der Fokus nicht nur auf der Endnutzerbedienbarkeit, sondern zudem auf der Effektivität von Systemen, die alleine mit automatischer Inhaltsanalyse arbeiten. Denn es gibt neben den KIS Ausgaben, auch Aufgaben aus den Bereichen der vollkommen automatischen Suche. Hierbei werden großen Mengen an Multimediateien verarbeitet, um bestimmte Ereignisse zu finden. Insbesondere bei Überwachungsvideos sollten Objekte automatisch erkannt werden und eine Liste mit gefundenen Szenen als Ergebnis geliefert werden. Ein Indikator, welcher für eine Messung herangezogen wird, ist die Genauigkeit dieser Ergebnisse. Um die Resultate dieser umfangreichen Evaluierung so repräsentativ wie möglich zu machen, wird immer viele Stunden Videomaterial herangezogen. Im Jahr 2010 waren es beispielsweise 400 Videostunden [55] [56]. Da mehrere Kategorien bzw. Tracks existieren, muss nicht jedes Team bei jeder Aufgabenart teilnehmen. Wie zuvor schon erwähnt können die Aufgaben in insgesamt drei verschiedene Aufgabenbereiche aufgeteilt werden. Im einzelnen sind das Suchaufgaben, bei denen die Systeme vollständig automatisch arbeiten, also keine menschliche Interaktion bei der Suche notwendig ist. Die zweite Art von Aufgaben sind manuelle Suchaufgaben, bei denen eine Person die Suchanfrage selbst formuliert und diese dem System als Parameter übergibt. Die dritte Kategorie sind interaktive Suchaufgaben, bei denen der manuelle Ansatz insofern erweitert wird, dass die gelieferten Ergebnisse vom Nutzer evaluiert werden und die Suchanfrage angepasst werden kann, sodass die neuen Ergebnisse genauer sind. Objekte, nach denen gesucht werden kann, können von ganz unterschiedlicher Natur sein. Beispielsweise kann nach ganz allgemeinen Eigenschaften gesucht werden, wie Bewegungen oder Farben. Dies kann jedoch auch expliziter sein, wie die Suche nach Gesichtern, Logos oder Texturen. Die Evaluierung der teilnehmenden Video Analyse Systemen erfolgt vollständig auf Basis deren gelieferten Ergebnisse, welche in einem einheitlichen Format sein müssen. Diese Ergebnisse werden dann abhängig vom Typ der Suchaufgabe entweder manuell bewertet oder von einem automatischen Bewertungssystem. [57]

In diesem Kapitel wurden verschiedene Methoden und Applikationen von Videobrowsing vorgestellt, sowie Möglichkeiten für deren Evaluierung. Im folgenden Kapitel wird erörtert, wie solche Systeme auf mobilen Geräten umgesetzt werden.

KAPITEL

4

Videonavigation auf Tablets und Smartphones

In den vorherigen Kapiteln wurden insbesondere Browser vorgestellt, die mit einem Desktop-Computer oder Laptop zu bedienen waren. Wenn es aber zum Thema Smartphones und Tablets kommt, kommen weitere Faktoren hinzu, die die Probleme noch verstärken. Aufgrund limitierter Bildschirmgröße müssen Kompromisse bei der Darstellung eingegangen werden, denn den Inhalt einfach zu verkleinern ist in vielen Fällen unmöglich. Die Interaktion muss für mobile Geräten mit Touch-Eingabe ebenso komplett überdacht werden, da Eingabeperipherie wie Maus und Tastatur nicht zur Verfügung stehen. Da aber heutzutage die Touchscreens mit einer hohen Präzision arbeiten, bieten sich somit ganz neue Möglichkeiten, was das Bedienkonzept betrifft. In diese Richtung weiter zu entwickeln erhält einen besonderen Stellenwert, da mittlerweile eine Vielzahl von verschiedenen Medien mobil konsumiert werden. Darunter sind beispielsweise Fernsehsendung, die in Echtzeit übertragen werden können, Filme und Serien, aber noch viel wichtiger ist die private Videosammlung. Hierbei zählen Videos, die der Nutzer selbst aufgenommen hat, sowie Videomaterial, dass direkt aus dem Internet auf das Gerät geladen wurde. Durch die Vielfalt ergibt sich auch, dass die Videodateien sich in ihren Eigenschaften stark unterscheiden. So sind die Aufnahmen von hoher bis niedriger Qualität, die Länge der Aufnahmen schwankt stark und auch die Aufnahmen werden sehr amateurhaft bis hochprofessionell produziert. Dies zusammen macht es umso schwieriger einen Videoplayer vorzuschlagen, welcher mit allen Arten sinnvoll zu bedienen ist. Ein weiteres Problem sind die teilweise stark eingeschränkten Ressourcen. Im Vergleich zu einem vollwertigen Heimcomputer, welcher in der Regel mit viel leistungsfähigeren Prozessoren und mehreren Gigabyte Arbeitsspeicher ausgestattet ist, können bei Smartphones und Tablets nur stromsparende Komponenten verbaut werden, um eine höher Akkulaufzeit zu erreichen und die Hitzeentwicklung zu minimieren. [19]

Im folgenden Kapitel werden die Besonderheiten aufgearbeitet, die es bei mobilen Geräten bezüglich Videobrowsing gibt und welche Faktoren bei der Gestaltung von mo-

bilen Applikationen beachtet werden sollten. Des Weiteren werden Lösungsvorschläge präsentiert, die als Motivation dienten, für die Implementierung des Prototyps, der im Rahmen dieser Arbeit entwickelt und evaluiert wurde.

4.1 Interface Design Guidelines

In den vorherigen Kapiteln wurde schon angeschnitten, das für mobile Geräte andere Design Regeln bezüglich deren Benutzeroberfläche und Interaktionsmöglichkeiten gelten. Im Jahr 2004, also noch bevor Smartphones und Tablets eine derart weitreichende Verbreitung erlangt haben, haben Gong und Tarasewich [59] diese Probleme genauer untersucht und einen Design-Guide veröffentlicht. Dabei haben sie versucht die sogenannten „Golden Rules of Interface Design“ von Shneidermann [58], welche im Jahr 1992 für Desktop-Computer entwickelt wurden, auf mobile Geräte zu übertragen. Diese insgesamt acht Regeln wurden um weitere sieben Regeln erweitert. Eine dieser Regeln ist, dass das Design ganz allgemein für kleine Bildschirme angepasst werden muss. Hierbei sollen beispielsweise Texte aus einer vorgefertigten Menge auswählbar sein, anstatt eine manuelle Texteingabe zu erfordern. Haptische Rückmeldungen vom System sollten ebenso Teil der Software sein. Was zusätzlich beachtet werden sollte ist, dass die Interaktion so einfach muss wie möglich, da komplizierte Aufgaben den Nutzern vom eigentlichen Inhalt ablenken. Am Beispiel eines Videoplayers kann eine besonders aufwändige Navigationsmethode den Nutzer derart fordern, dass dieser vom Video nur wenig aufnimmt und so das Ziel leicht verfehlt werden kann. Ein weiterer Punkt ist der, dass die Informationen, die die Applikation bereitstellt, limitiert werden müssen, sodass der Nutzer auch hier nicht überfordert wird. Das bedeutet im Detail, dass am Beispiel des Videoplayers nur relevante Informationen, wie die aktuelle Position oder sonstige Inhalte, die für die explizite Interaktion notwendig sind, angezeigt werden sollten. Einer der wichtigsten Punkte ist jedoch, dass die Benutzeroberfläche von Grund auf ansprechend gestaltet werden muss, unabhängig vom Zweck der Applikation. [59]

4.2 Multi-Touch Bildschirme und Gesten

Berührungsempfindliche Bildschirme ermöglichen eine besonders einfache Kommunikation mit einem Computer. Die direkte Interaktion ist für die meisten Nutzer, insbesondere Anfänger sehr ansprechend, da man nicht erst den Umgang mit einer Eingabeperipherie üben muss. Daraus ergibt sich eine steile Lernkurve. Problematisch wird es nur, wenn der Nutzer eine Benutzeroberfläche bedienen muss, die eigentlich für den Gebrauch mit einer Maus und Tastatur entwickelt wurde. Aktionen wie ein Rechts-Klick werden hier

zu einem großen Hindernis. Auch die Größe der Knöpfe spielten hier eine nicht unwesentliche Rolle bezüglich des Komforts [60]. Anfangs war dies auch kein großes Problem, da Geräte mit Touchscreens in der Regel nicht in privaten Haushalten zu finden waren. Vor allem Bankautomaten, Zugticketautomaten und Informationskiosk setzten auf dieses Konzept. Dadurch wurde die Langlebigkeit dieser Maschinen erheblich verbessert, denn auf beweglichen Teile wie Mäuse konnte verzichtet werden [61]. Generell gibt es berührungsempfindliche Bildschirme schon seit Jahrzehnten, jedoch haben sie erst durch das Apple iPhone derart an Popularität gewonnen. Die Steuerung ist enorm präzise, weshalb die Applikationen keinerlei Hardware-Knöpfe mehr benötigen. Virtuelle Tastaturen und schnell durchsuchbare Listen wurden damit massentauglich. Multi-touch gibt den Bildschirmen die Fähigkeit auf Eingaben zu reagieren, die mit mehr als nur einem Finger gemacht werden. Viele Gesten wurden somit erst möglich, die heutzutage allgegenwärtig sind. Diese sind beispielsweise Wischgesten, Pinch-Gesten, die beim vergrößern bzw. verkleinern von Objekten zum Einsatz kommen, sowie Rotationsgesten, um Fotos zu drehen. Die Objekte können daher direkt manipuliert werden, ohne dass der Nutzer sich Eingabebefehle wie Tastenkombinationen merken muss [62]. Die weitverbreiteste Technologie, die bei den heutigen Smartphones und Tablets zum Einsatz kommen, sind sogenannte kapazitive Touchscreens, die mit Hilfe von elektrischer Spannung auf dem Glas, die durch die Finger unterbrochen wird, die exakte Position feststellen können. [63] Generell gibt es heutzutage sehr viele verschiedene Wischgesten, um in einem mobilen Betriebssystem oder auch einer einzelnen Applikation zu navigieren. Alleine Apple stellt insgesamt sechs zweidimensionale Standardgesten den Entwicklern in einer Programmierschnittstelle zur Verfügung. Dazu zählen beispielsweise die schon genannten Pinch- und Rotationsgesten. In der Abbildung 4.1 sind alle Gesten einzeln dargestellt. Bis auf die Pinch-Geste sind alle Gesten mit einem Finger durchzuführen, jedoch kann dies so angepasst werden, dass eine Geste mehrere Finger benötigt. Beispielsweise ist die Rotationsgeste eine Drag-Geste, die mit zwei Fingern gemacht wird. Um Objekte anzutippen kann eine einfache Tap-Geste benutzt werden, die auch so eingestellt werden kann, dass zwei oder mehrere Finger benutzt werden müssen. Dies ist beispielsweise der Fall, wenn man einen Text auf einer Internetseite vergrößern und zentrieren möchte. Die Touch-and-Hold-Geste kann mit dem Rechts-Klick auf einem Desktop-Computer verglichen werden. Der Nutzer berührt ein Element auf dem Bildschirm und bleibt so lange mit dem Finger auf dem Bildschirm, bis sich ein Kontextmenü öffnet. Dies kann beispielsweise benutzt werden, um einen Text zu markieren. Zwei Gesten, die sich sehr ähnlich, jedoch nicht das gleiche sind, sind die Swipe-Geste und die Flick-Geste. Die Swipe-Geste wird in der Regel verwendet, wenn eine genauere und kontrollierte Navigationsinteraktionen gemacht werden müssen. Dies ist zum Beispiel der Fall, wenn man innerhalb einer Liste eine Zelle seitlich mit einem Wisch verschiebt, sodass eine Löschfunktion zum Vorschein kommt.

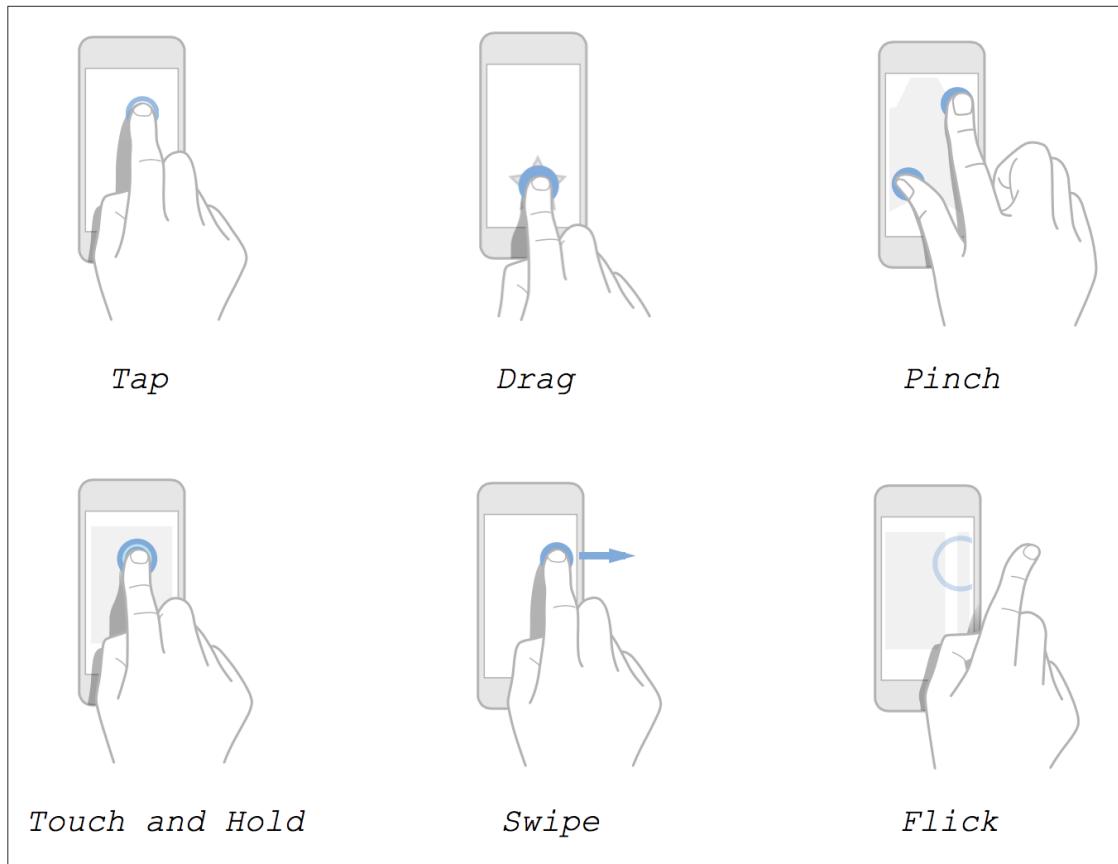


Abbildung 4.1: Standardgesten für iOS [15]

Ein weiterer Anwendungsfall ist das Betrachten einer Reihe von Fotos, bei denen man mit einem horizontalen Wisch von einem Foto zum nächsten gelangt. Die Flick-Geste hingegen wird meist schneller und unpräziser durchgeführt. Mit einer ruckartigen Bewegung kann eine Beschleunigung einer Liste gestartet werden, bei der man nicht genau vorhersehen kann, an welcher Stelle man sich befinden wird, wenn die Bewegung von alleine zum Stillstand gekommen ist. Aus diesem Grund können solche Interaktionen mit einer Tap-Geste unterbrochen werden, sodass das durchblättern der Liste gestoppt wird. [15]

Der Prototyp des Videoplayers, der im Rahmen dieser Arbeit entwickelt wurde, basiert auf einer Flick-Gesten Navigation. In den folgenden Kapiteln werden jedoch zuerst anderen Arten der Videonavigation auf mobilen Geräten aufgearbeitet.

4.3 Standardvideoplayer

Jedes Smartphone und Tablet kommt mit einem vorinstallierten Standardvideoplayer beim Kunden an. Diese unterscheiden sich im Kontext von Bedienung und Benutzeroberfläche in der Regel kaum voneinander. Üblicherweise enthält die Oberfläche einen Start-Knopf, bzw. Pause-Knopf, der sich je nach Wiedergabestatus verändert, zwei Knöpfe um jeweils in die vorherige, bzw. nächste Datei zu wechseln und einen Lautstärkeregler. Um aber innerhalb eines Videos zu navigieren wird eine Suchleiste angezeigt, die je nach zeitlicher Position in Relation zum Video einen Punkt markiert, welcher sich auch verschieben lässt. Im Falle des Standardplayers der iOS Geräte iPhone, iPod und iPad kann der Nutzer beim Drücken des Positionsknopfes einem vertikalen Wisch nach unten durchführen und dadurch die Bewegung verlangsamen, sodass der Vorlauf, bzw. Rücklauf verfeinert ist. Die Position an der Oberseite des Bildschirmes ist aber aus benutzertechnischer Sicht nicht optimal, da der Inhalt mit der Hand verdeckt wird [64]. Der *VLC Videoplayer* von VideoLAN ist stark an das Bedienkonzept des Standardplayers angelehnt. Die folgende Abbildung 4.2 zeigt den Videoplayer auf einem iPad mit 9,7 Zoll

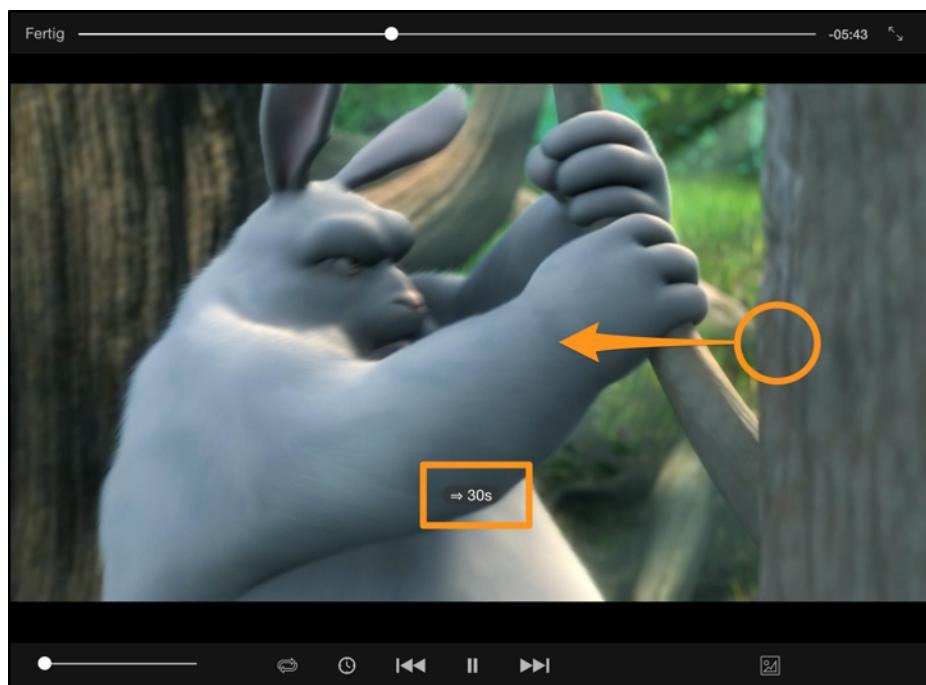


Abbildung 4.2: VLC Player auf einem iPad

Bildschirm. Das Besondere an dieser Applikation ist jedoch nicht das Bedienkonzept. Der VLC Player legt den Fokus auf eine hohe Kompatibilität zu einer Vielzahl von Audio

und Video Formaten. Dennoch versuchen solche standardisierten Videoplayer mit Hilfe von Extrafunktionen die Navigation etwas zu vereinfachen und sich zu differenzieren. Es gibt unterschiedliche Ansätze Wischgesten für die Navigation innerhalb eines Videos oder einer Videobibliothek zu verwenden. Häufig werden Methoden verwendet, um mit einer horizontalen Wischbewegen im Video entweder nach vorne oder nach hinten zu springen. Dadurch verliert der Benutzer offensichtlich Informationen darüber, was zwischen den beiden Szenen passiert. Insbesondere wenn das Ziel ist, den gesamten Inhalt eines Videos zu erfassen, ohne sich den Inhalt in einfacher Geschwindigkeit anzusehen, kommen aktuelle Videoplayer an ihre Grenzen. Als Beispiel einfacher Wischfunktionen ist der VLC Player für iOS in der Version 2.6.6. Mit einem horizontalen Wisch von rechts nach links springt der Benutzer um fixe 30 Sekunden nach vorne im Video. Problematisch hierbei ist, dass der Benutzer keine Möglichkeit hat das Intervall anzupassen, um sich ggf. schneller bzw. langsamer im Video zu bewegen. Des Weiteren ist die fixe Sprunglänge suboptimal für besonders lange oder kurze Videos. [65]

Im folgenden Kapitel werden Prototypen und Konzepte von Videobrowsern vorgestellt, die sich als Alternative zu klassischen Videoplayern positionieren und für den Einsatz auf mobilen Geräten bezüglich Bedienung und Darstellung optimiert sind.

4.4 Videobrowsing auf mobilen Geräten

Dieses Kapitel widmet sich den Videobrowsern, die speziell für mobile Geräten mit Touch-Eingabe optimiert sind. Der Fokus liegt hier insbesondere auf die Seite der Player, die entweder den Inhalt alternativ darstellen oder sich in punkto Navigation von dem klassischen Suchleistenprinzip abwenden.

Der *ThumbBrowser* von Hudelst et al. [16] ist ein Videoplayer, der sich die natürliche Haltung des Tablets im Querformat in zwei Händen zum Vorteil macht, da dann die beiden Daumen des Nutzers frei beweglich sind. Das Querformat wird von den Nutzern besonders häufig gewählt, da bedingt durch die Aufnahme der Inhalt des Videos so den Bildschirmplatz am besten ausnutzt. Insgesamt enthält der ThumbBrowser zwei Menüs, die jeweils von einem Daumen gesteuert werden. In Abbildung 4.3 ist die Benutzeroberfläche zu sehen, wie sie von einer Person bedient wird. Die beiden Menüs sind nur sichtbar, wenn der Bildschirm berührt wird. Des Weiteren positionieren sie sich immer an der Stelle, an welcher man mit dem Gerät in Kontakt kommt. Der rechte Daumen aktiviert die Navigationsleiste, welche vertikal ausgerichtet ist. Das obere Ende repräsentiert den Anfang des Videos und das unter Ende den Schluss. An der Position des Daumen befindet sich ein Vergrößerungsglas, welches ein Vorschaubild der Position zeigt, an die die Navigationsleiste markiert ist. Dieses Prinzip ist auch schon vom YouTube Videoplayer bekannt [66]. Wenn man an die markierte Stelle springen möchte,



Abbildung 4.3: ThumbBrowser in Benutzung [16]

dann hebt man den Daumen an und das Menü verschwindet wieder. Falls man jedoch die Aktion abbrechen möchte, ohne sich im Video zu bewegen, so kann man mit dem Schieberegler entweder an das obere oder untere Ende fahren und dann erst den Daumen vom Bildschirm lösen. Des Weiteren bietet die Oberfläche noch ein Textfeld, welches die zeitliche Position angibt, von der die angezeigte Vorschau stammt. Wenn der Nutzer den linken Daumen auf dem Bild platziert, dann öffnet sich an genau dieser Stelle ein Torten-Menü, welches den Namen aufgrund der kuchenartigen Form hat. Dieses Menü bietet verschiedene Menüpunkte, wie einen Stern, der eine bestimmte Stelle im Video markiert. Des Weiteren gibt es rechts und links jeweils einen Knopf, mit dem man im Schnelldurchlauf nach vorne und zurück spulen, sowie eine Lupe, mit dem der Nutzer in die Navigationsleiste hineinzoomen kann, sodass wiederum mit dem rechten Daumen feiner navigiert wird. Der ThumbBrowser beschränkt sich momentan auf die erweiterte Navigation, ohne dabei automatische Inhaltsanalysen zu machen. Da für diese Demo Applikation keine Benutzerstudie durchgeführt wurde, lässt sich keine Aussage darüber treffen, ob sich diese Methode im Alltag als tauglich beweisen kann. [16]

Der Videobrowser *PocketDRAGON* von Karrer et al. [17] basiert auf der Annahme, dass Menüs, die über dem Video liegen, vom eigentlichen Inhalt ablenken. Auf Basis dieser Annahme implementierten die Autoren einen Videoplayer, welcher keine sichtbaren Menüs besitzt, sondern mit Hilfe von Objekterkennung den Nutzer die Möglichkeit gibt, durch das Video zu navigieren. Das eigentliche System hinter diesem mobilen Videobrowser ist *DRAGON*. Mit Hilfe von verschiedenen Algorithmen werden die Objekte, die sich in einem Video befinden, erkannt und deren Bewegungslinien analysiert. Der Nutzer kann somit auf der Desktop-Version dieses Browsers mit der Maustaste ein beliebiges Objekt festhalten und es entlang des Bewegungsablaufes ziehen, was dann einen Vor- bzw. Rücklauf im Video zur Folge hat. Dieses Konzept wurde nun auf ein Smartphone übertragen, da der Nutzer so direkt mit den Objekten interagieren kann. Da die Leistung

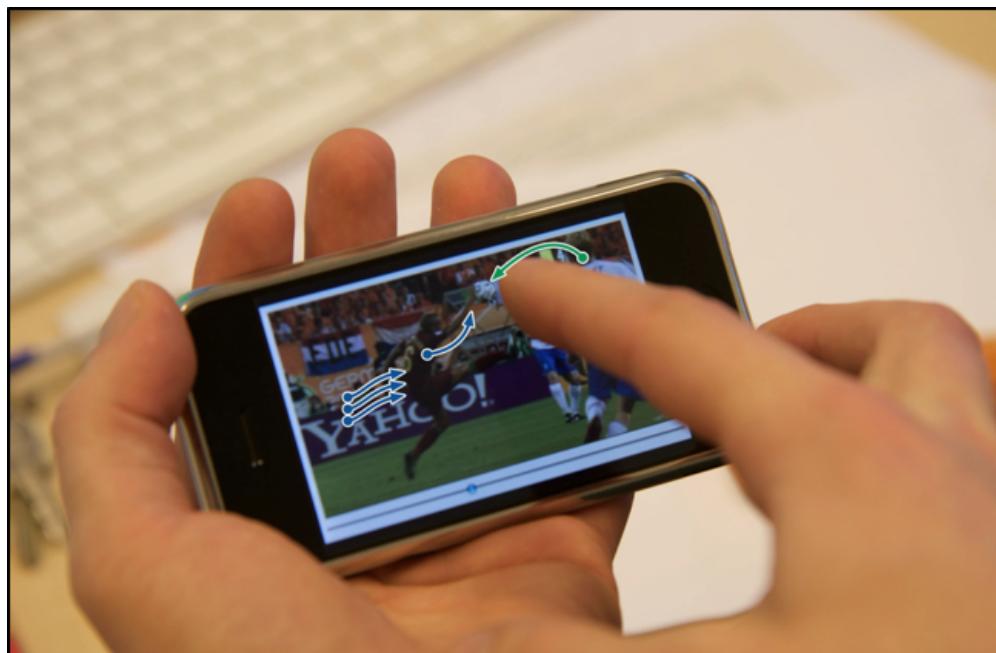


Abbildung 4.4: PocketDRAGON mit Fußballszene [17]

der mobilen Geräte nicht ausreicht, werden die besonders rechenintensiven Algorithmen extern auf einem angeschlossenen Server ausgeführt. Da das iPhone mittels Netzwerk-kommunikation die Daten mit dem Server austauscht, kommt es zu Verzögerungen bei der Navigation. Diese konnte jedoch durch einen eigens für diesen Zweck entwickelten Algorithmus, welcher flexibel die einzelnen Frames zwischenspeichert und zur Vorschau lädt, minimiert werden. Des Weiteren werden die Videoframes in verschiedenen Auflösungen vorgeladen, sodass bei groben Bewegungen die Frames gezeigt werden, welche

eine niedrige Auflösung haben und bei langsamem, detaillierteren Bewegungen, diese mit hoher Auflösung. Abbildung 4.4 zeigt die Bedienung des PocketDRAGON, mit einer Szene aus einem Fußballspiel. Auf dem Bild ist gut zu erkennen, wie der Nutzer mit dem Finger den Ball festhält und ihn genau an die Stelle seines Interesses bewegt. Die vorgestellte Version ist jedoch nicht in der Lage diese Navigationsmethode über eine längere Sequenz des Videos durchzuführen, sondern eignet sich nur, um Szenen mit kürzerer Länge in detaillierter Weise zu betrachten. Eine klassische Navigationsleiste für das schnelle Bewegen wird zusätzlich unterhalb angezeigt. Das Ziel dieses Projektes war es nicht eine sogenannte Standalone Applikation zu schaffen, welche Marktreife hat, sondern festzustellen, ob das Bedienkonzept auf Smartphones und Tablets sinnvoll ist. Eine Benutzerstudie wurde nicht durchgeführt, weshalb sich keine gültige Aussage darüber treffen lässt.

Der *Keyframe Navigation Tree* von Hudelist et al. [18] ist ein Prototyp für einen Videoplayer mit einer erweiterten Navigationsleiste. Diese besteht im Prinzip aus drei einzelnen Leisten, die oberhalb miteinander verbunden sind. Die oberste Navigationsleiste zeigt eine Reihe von sogenannten Keyframes, also Bilder, die aufgrund ihrer Position und Qualität besonders viele Informationen über den Inhalt des Videos liefern. Diese werden jedoch nicht vollständig dargestellt, sondern als vertikale Ausschnitte, die alle direkt aneinander gelegt sind. Von links nach rechts ergibt sich somit eine Reihe dieser Keyframes, welche chronologisch beginnend vom Anfang des Videos extrahiert wurden. Die zweite Navigationsleiste darunter funktioniert nach dem selben Prinzip. Hier sind die einzelnen Vorschaubilder etwas größer, was einen tieferen Einblick in den Inhalt zulässt. Das hat zufolge, dass diese Übersicht nur einen Ausschnitt des gesamten Videos zeigt. Die unterste Navigationsleiste zeigt dann die Vorschaubilder im vollen Umfang an und ist daher am genauesten. Abbildung 4.5 zeigt eine Übersicht dieser Zeitleiste. Die rote durchgezogene Linie markiert die Stelle, an der das Video gerade abgespielt wird. Da die verschiedenen Leisten immer den selben Inhalt anzeigen, jedoch in unterschiedlicher Detailstufe, ist der rote Markierer für alle Leisten gültig. Mit dem Finger können alle drei Bereiche unabhängig voneinander bewegt werden, was das Video nach vorne, bzw. hinten navigieren lässt. Sobald der Finger vom Bildschirm gelöst wird, spielt das Video an dieser Stelle dann ab. Eine Benutzerstudie welche im Rahmen dieses Projekts durchgeführt wurde, kam zu dem Schluss, dass die Teilnehmer die Navigation als viel einfacher und intuitiver wahrgenommen haben, die gestellten Suchaufgaben schneller lösen konnten und auch mehr Spaß hatten. Insgesamt haben nur drei von den 20 Kandidaten den Standardvideoplayer bevorzugt, gegen den die Vergleichsmessung durchgeführt wurde.

Der *Mobile ZoomSlider* von Hürst et al. [19] verfolgt den Ansatz eine Navigation ohne fixer Zeitleiste zu ermöglichen. Hierbei ist der Gedanke, dass der Nutzer überall auf den Bildschirm tippen kann und sich an dieser Stelle die Funktion einer Navigationsleiste



Abbildung 4.5: Keyframebasierter Navigationsbaum [18]

aktiviert. Der große Vorteil dabei ist, dass kleine Knöpfe auf einem Schieberegler nicht mühsam angepeilt werden müssen. Die horizontale Bewegung bestimmt, ob im Video nach vorne oder hinten navigiert wird und die vertikale Position des Eingabestifts gibt an, wie schnell man sich im Video bewegt. Daher auch der Name ZoomSlider, da die vertikale Bewegung die Zeitleiste entweder vergrößert oder verkleinert. Als zusätzliche Funktion wurden sogenannte *Speedborders* implementiert. Das bedeutet, dass der Nutzer an die rechte oder linke Kante drücken kann, um einen Schnellvorlauf, bzw. Rücklauf zu aktivieren. Auch wurde wieder die vertikale Position des Stiftes mit einbezogen, sodass der Schnelllauf weiter oben langsamer ist, verglichen zum unteren Bereich. Damit der Nutzer immer weiß, welche Auswirkungen seine Aktionen haben, werden unterschiedliche visuelle Feedbacks gegeben. In Abbildung 4.6 sind beispielsweise die Symbole zu

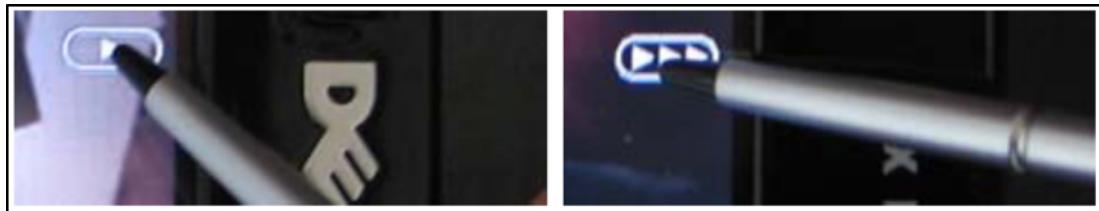


Abbildung 4.6: Mobile ZoomSlider Schnelldurchlauf durch Speedborders [19]

sehen, welche beim Schnelldurchlauf eingeblendet werden. Hier ist zusätzlich noch an-

zumerken, dass die Symbole immer an der Stelle eingeblendet werden, an der der Stift den Bildschirm berührt. Im Zuge der Implementierung wurde im Anschluss der Mobile ZoomSlider mittels Benutzerstudie evaluiert. Zu Beginn haben die Teilnehmer eine kurze Einführung in die Handhabung des Videoplayers bekommen. Ziel der Studie war es herauszufinden, wie gut die Probanden die Navigationsmethode verstehen und ob sie damit Suchaufgaben lösen könnten. Des Weiteren wurden die subjektiven Eindrücke festgehalten. Die Ergebnisse waren, dass die Bedienbarkeit als natürlich und positiv empfunden wurde, sowie eine steile Lernkurve verzeichnet werden konnte. Die Suchaufgaben konnten ebenfalls gut gelöst werden.

Das nächste Beispiel eines mobilen Videobrowsers ist dafür vorgesehen, dass man sich sowohl innerhalb eines Videos, als auch zwischen verschiedenen Videos navigieren kann. Der *Wipe'n'Watch* Player von Huber et al. [20] ist eine iPhone Applikation, die speziell für den Fall von eLearning Material entwickelt wurde. Das bedeutet, dass die verwendeten Videomaterialien in der Regel Aufzeichnungen von Universitätsvorlesungen sind. Diese zeigen nur die präsentierten Folien als visuelles Objekt und die Audiospur ist der Kommentar des Vortragenden. Anstatt aber eine Zeitleiste zu benutzen, springt man immer zum nächsten Keyframe. Dies geschieht mit einem horizontalen Wisch. Die Richtung der Bewegung gibt an, ob man sich nach vorne oder nach hinten bewegt. Da die Videos nur die Vorlesungsfolien anzeigen, sind die Keyframes also immer der Beginn einer neuen Folie. So springt man also beim navigieren immer von einer Folie zur nächste im Video und die Audiospur passt sich dementsprechend an. Somit ist es möglich das Video wie eine PDF Datei zu lesen und zu navigieren. Beim *Wipe'n'Watch* Player werden Videos mit Hyperlinks verwendet, die auch Hypervideos genannt werden. Das bedeutet, dass an verschiedenen Stellen im Video Links zu anderen Videos sind, die semantisch miteinander zusammenhängen. Sobald eine Folie angezeigt wird, welche also inhaltlich mit einem oder mehreren Videos zusammenhängt, wird ein kleines Symbol eingeblendet. In diesem Fall kann der Nutzer mit einem vertikalen Wisch nach oben eine Übersicht mit allen dieser Videos öffnen. Abbildung 4.7 zeigt, wie diese Interaktion auf einem Gerät aussieht. Auf dem Bild a) sind zwei referenzierte Videos in der Übersicht enthalten, das ist zu erkennen, da sie sich in der grauen Box befinden. Mit einem Klick auf eines der Videos wird dieses abgespielt. Mit einem Wisch nach unten wird die Übersicht wieder geschlossen. Ein langer Druck auf den Bildschirm öffnet den Verlauf der letzten Videos, was in Abbildung 4.7b) zu sehen ist. Eine Benutzerstudie mit 44 Probanden hat gezeigt, dass komplexe Suchaufgaben mit den *Wipe'n'Watch* Player schneller gelöst werden konnte, verglichen zu einem nur leicht veränderten Standardvideoplayer. Zudem wurde ein hohes Verständnis für die Zusammenhänge der Videos geschaffen, sowie positives zur Benutzerfreundlichkeit beigetragen. [20]

Ein Videobrowser, welcher auf einem ähnlichen Navigationsprinzip aufbaut, ist der

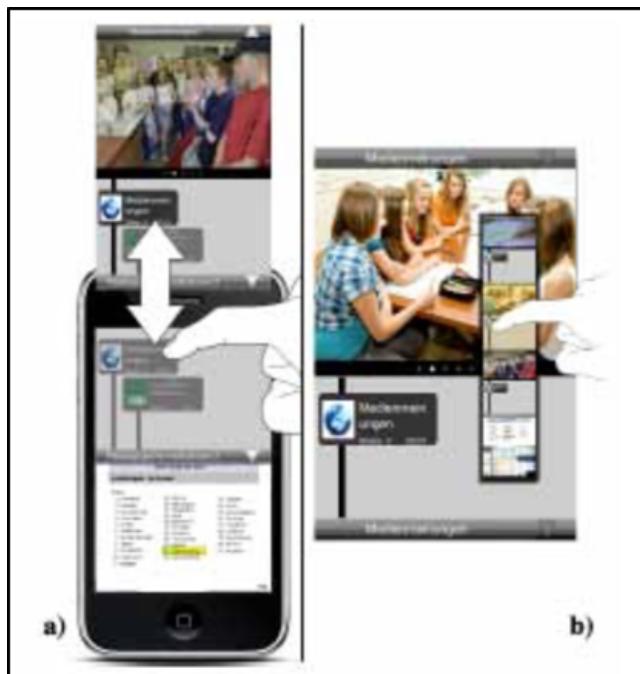


Abbildung 4.7: Wipe'n'Watch Player mit inter-Video Navigation [20]

SwiPlayer von Schöffmann et al. [21]. Auch hier werden horizontale und vertikale Wischgesten benutzt, um sich innerhalb eines Videos, bzw. zwischen verschiedenen Videos zu bewegen. Der Videoplayer ist jedoch nicht für eine bestimmten Videotyp ausgelegt, sondern für jede Art gleichermaßen geeignet. Das Bedienkonzept geht dabei so vor, dass eine Navigationsleiste vollkommen ausgelassen wurde und nur die direkte Interaktion mit dem Video im Vordergrund steht. Ähnlich wie bei einer Fotoalben Applikation auf dem Smartphone oder Tablet kann man das angezeigte Bild nach rechts oder links wegwischen, um zum nächsten bzw. vorherigen Bild zu gelangen. Nur dass in diesem Fall keine Bilder betrachtet werden, sondern der Inhalt eines Videos. Wenn der Nutzer also den Bildschirm berührt, dann pausiert das Video, bis es wieder losgelassen wurde. In Abbildung 4.8 (links) ist zu sehen, wie das Video von rechts nach links geschoben wird. Währenddessen bewegt sich von rechts die neue Position ins Bild, an der das Video dann weitergeht. Um eine Verfeinerung der Navigation zu ermöglichen, ist die Sprungweite immer prozentual von der Videolänge abhängig. Dieser Prozentwert variiert je nach vertikaler Position des Fingers. Wie in Abbildung 4.8 (rechts) zu sehen ist, gibt es insgesamt fünf verschiedene Werte, die angenommen werden können. Grobe Sprünge von 8% können oben erreicht werden, bis zu feineren 0,5% am unteren Bildschirmrand. Der Nutzer hat jederzeit die Möglichkeit die Aktion abzubrechen, indem er die Bewegung

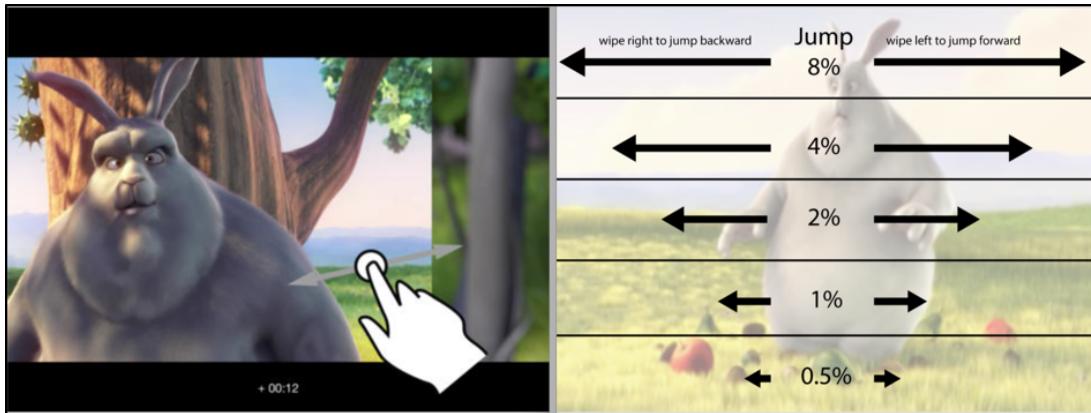


Abbildung 4.8: SwiPlayer intra-Video Navigation und Granularität [21]

nicht vollständig ausführt, den Finger also anhebt und der Videoplayer sich somit wieder auf die ursprüngliche Position ausrichtet und die Wiedergabe fortsetzt. Falls das Video von Beginn an im pausierten Modus war, dann würde das Video nicht sofort laufen, sondern muss manuell mit einem 2-Finger Tipp die Wiedergabe starten. Ein vertikaler Wisch von unten nach oben hat zur Folge, dass das nächste Video in der Sammlung gespielt wird. Die letzte Position des vorherigen Videos wird jedoch gespeichert, sodass der Nutzer dort fortfahren kann. Somit ist es möglich, ähnlich wie bei dem ThumbBrowser [16], den SwiPlayer ausschließlich mit den Daumen zu bedienen. Für eine Evaluierung wurde im Rahmen des Projektes eine Benutzerstudie durchgeführt. Dort wurde der Prototyp des Videoplayers mit dem Standardvideoplayer des iPads verglichen. Insgesamt 24 Probanden haben mehrere Zielsuchaufgaben gemacht, sowohl mit dem SwiPlayer, als auch mit dem Standardplayer. Die Ergebnisse waren, dass die Suchaufgaben mit dem wischgestenbasierten Player etwas schneller fertiggestellt werden konnten. Die subjektive Wahrnehmung der Nutzer war, dass die Navigationsmethode sehr schnell zu erlernen sei und 87% der Teilnehmer diesen Videobrowser für das Erfüllen der Aufgaben bevorzugten.

In diesem Kapitel wurden nun einige Ansätze vorgestellt, wie Videonavigation bzw. Videobrowsing auf Smartphones und Tablets umgesetzt werden kann. Die verschiedenen Prototypen verwendeten teilweise Methoden aus Video Retrieval, beispielsweise, um Keyframes aus dem Video zu identifizieren, oder um den Bewegungsverlauf von Objekten aufzuzeichnen. Jedoch war eine Kombination mit einer veränderten Darstellung des Inhaltes oder Navigationshilfen wie Gesten immer notwendig, um einen tatsächlichen Vorteil zu generieren. Die gewonnenen Erkenntnisse, insbesondere des SwiPlayers waren Ideengeber für den Prototypen, der im Rahmen dieser Arbeit umgesetzt wurde.

In den vorherigen Kapiteln wurden verschiedene Navigationsmethoden von Videoplayer Software vorgestellt. Diese waren sowohl aus dem Bereich der Desktop-Computer, sowie mobile Geräte wie Smartphones und Tablets. Letztere bieten mit ihren präzisen Touchscreens besondere Möglichkeiten. Im Rahmen dieser Arbeit wurde ein Prototyp eines Videoplayers entwickelt, welcher ausschließlich mit Wischgesten den Nutzer durch ein Video navigieren lässt. Im weiteren Verlauf der Arbeit wird dieser als *Flickplayer* bezeichnet. Der Name Flick kommt aus dem Englischen und bedeutet so viel wie etwas schnippen oder schnellen lassen. Insofern ist das passend, da dies die Bewegung beschreibt, die bei der Navigation angewendet werden muss. In diesem Kapitel wird der gesamte Entwicklungsprozess des Prototypen beschrieben. Beginnend von der Spezifikation, bis hin zu Details bezüglich der Implementierung.

5.1 Spezifikationen

Bevor mit der Implementierung begonnen werden konnte, musste zuerst genau ausgearbeitet werden, wie das Navigationskonzept funktionieren und die Benutzeroberfläche gestaltet werden sollte. Die Einzelheiten zur Konzeptionsphase werden deshalb in diesem Kapitel aufgearbeitet.

5.1.1 Anforderungsanalyse

Wischgestenbasierte Videoplayer haben oft das Problem, dass in unterschiedlicher Weise von einer Szene in die nächste gesprungen wird. Dabei wird ein Teil des Inhalts oft übersprungen, da der Nutzer diesen gar nicht zu sehen bekommt. Als Grundlage wurde der SwiPlayer [21] genommen, da hier im Rahmen einer Nutzerstudie herausgefunden wurde, dass das Bedienkonzept bei der Navigation unterstützte und sehr positiv von den Probanden bewertet wurde. Jedoch gab es genau hier die Limitierung, dass zwischen

den einzelnen Szenen der Inhalt übersprungen werden konnte. Dieses Problem sollte mit dem Flickplayer gelöst werden. Des Weiteren sollte die Benutzeroberfläche sehr einfach gestaltet werden, sodass eine Ablenkung durch viele Bedienelemente verhindert werden kann. Wischgesten eignen sich sehr gut, da somit auf Knöpfe verzichtet wird. Der Flickplayer sollte zudem in der Lage seine, eine Navigation sowohl nach vorne im Video, sowie rückwärts zuzulassen.

5.1.2 Navigationskonzept

Beim Navigationskonzept werden horizontale Wischgesten benutzt, um sich innerhalb eines Videos zu bewegen. Dabei werden jedoch keine sprunghaften Bewegungen auf der Zeitleiste gemacht, sondern die Wiedergabegeschwindigkeit des Videos dynamisch verändert. Standardmäßig werden Videos mit einer Geschwindigkeit von 100% abgespielt. Da auf eine Navigationsleiste bewusst verzichtet werden sollte, kann der Nutzer die Wiedergabegeschwindigkeit auf bis zu 32-fach bringen, also 3200%, um sich schnell im Video bewegen zu können. Der große Vorteil hierbei ist, dass man so von trotzdem beschleunigter Navigation dennoch einen Eindruck vom Inhalt des Videos bekommt. Das eigentliche Konzept hat starke Parallelen zur Navigation innerhalb von Listen auf Smartphones. Anstatt von Seiten, zwischen denen umgeblättert wird, kann der Inhalt festgehalten werden und schwungartig nach oben oder unten bewegt werden. Diese Navigationsart wird auch Scrolling genannt. Die dafür nötige Fingerbewegung ist eine sogenannte Flick-Geste. Die unten gezeigte Abbildung 5.1 veranschaulicht dieses Scrolling innerhalb eines Dokuments. Die einzelnen Seiten werden übergangslos aneinandergehängt, sodass der Nutzer mit einer Wischbewegung schnell über mehrere Seiten navigieren kann [22]. Bei dem Prototyp des Flickplayers wird daher die Wiedergabegeschwindigkeit des Videos angepasst. Um den Übergang von der 1-fachen Geschwindigkeit, bis hin zum Maximum, also 32-fachen Geschwindigkeit nicht ruckartig zu machen, gibt es mehrere Zwischenstufen als Übergänge. Diese sind 2-fach, 4-fach, 8-fach und 16-fach. Wie bei dem Scrolling eines Dokuments hat ein stärkerer Wisch einen größeren Effekt, als ein leichter, bzw. langsamerer Wisch. Die Übergänge zwischen den Abspielgeschwindigkeiten sollten somit für den Nutzer als flüssig wahrgenommen werden. Die gleichen Intervalle wurden ebenso für das Rückwärtsabspielen übernommen, sodass ein einheitliches Konzept entsteht. Ein Wisch von links nach rechts hat somit zur Folge, dass sich das Video rückwärts abspielt, bis hin zu einer Geschwindigkeit von minus 32-fach. In Abbildung 5.2 wird das Navigationskonzept nochmals grafisch dargestellt. Zu sehen ist ein Finger, welche die horizontale Bewegung andeutet, sowie die verschiedenen Stufen der Abspielgeschwindigkeit in beide Richtungen. Um von einer höheren Abspielgeschwindigkeit wieder in den Ausgangszustand zu gelangen, hat der Nutzer verschiedene Möglichkeiten. Da bei einem Wisch von

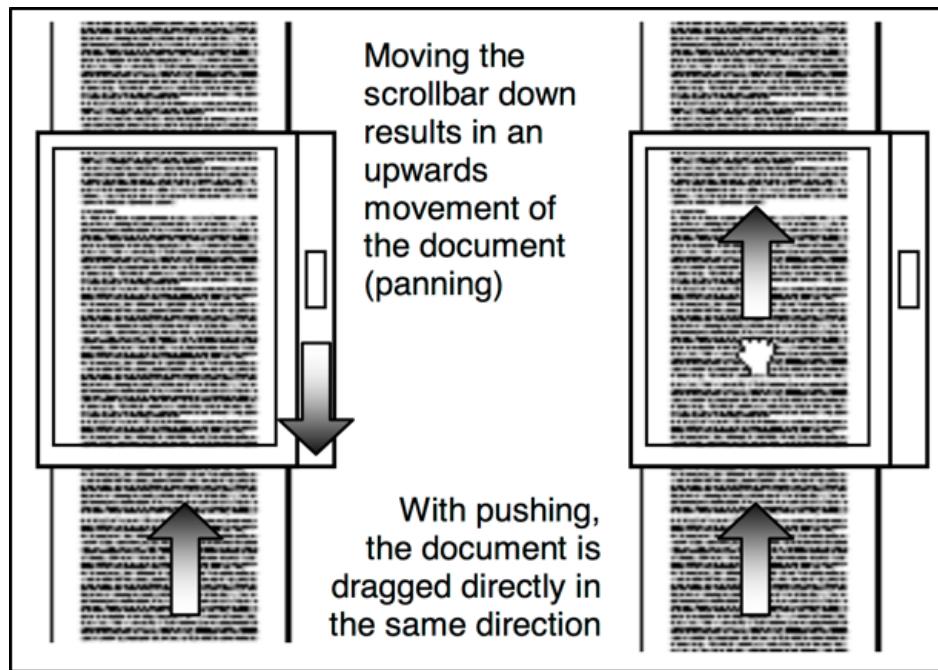


Abbildung 5.1: Navigation innerhalb von Textdokumenten [22]

rechts nach links die Geschwindigkeit zunimmt, baut diese sich im Laufe der Zeit mit einer linearen Easingfunktion wieder ab. Weitere Wischgesten in die selbe Richtung lassen den internen Wert weiter erhöhen. Dies ist so lange der Fall, bis das Maximum erreicht wird. Die Intervalle sind zwischen den Geschwindigkeiten alle exakt gleich groß, sodass im Fall des Maximums der Wert linear abnimmt und so alle Geschwindigkeitsstufen gleich lang aktiv sind. Eine lineare Easingfunktion eignet sich hierbei am besten, da ein gleichmäßiges Verlangsamten am natürlichen empfunden wird. Diese Annahme konnte durch nicht-dokumentierte Tests mit Benutzern bestätigt werden. Ein Wisch in die entgegengesetzte Richtung mindern diesen Wert schneller, sodass dieser irgendwann negativ wird und das Video rückwärts abgespielt wird. Der schnellste Weg, um wieder in die Ausgangssituation der 1-fachen Geschwindigkeit zu gelangen, ist ein einfacher Fingertipp auf den Bildschirm. Der intern verwendete Wert wird somit direkt auf null gesetzt. Das ist insbesondere hilfreich, wenn man versucht eine bestimmte Szene zu finden und mit hoher Geschwindigkeit durch das Video navigiert. Sobald man glaubt die Szene gefunden zu haben, stoppt man das schnelle Abspielen und kann so in 1-facher Geschwindigkeit die Szene genauer untersuchen. Wenn man am Ende des Videos angelangt ist, gelangt man automatisch wieder zum Anfang des Videos. Umgekehrt ist dies auch möglich, denn wenn der Nutzer ein Video startet und direkt zum Ende des Videos gelangen möchte,

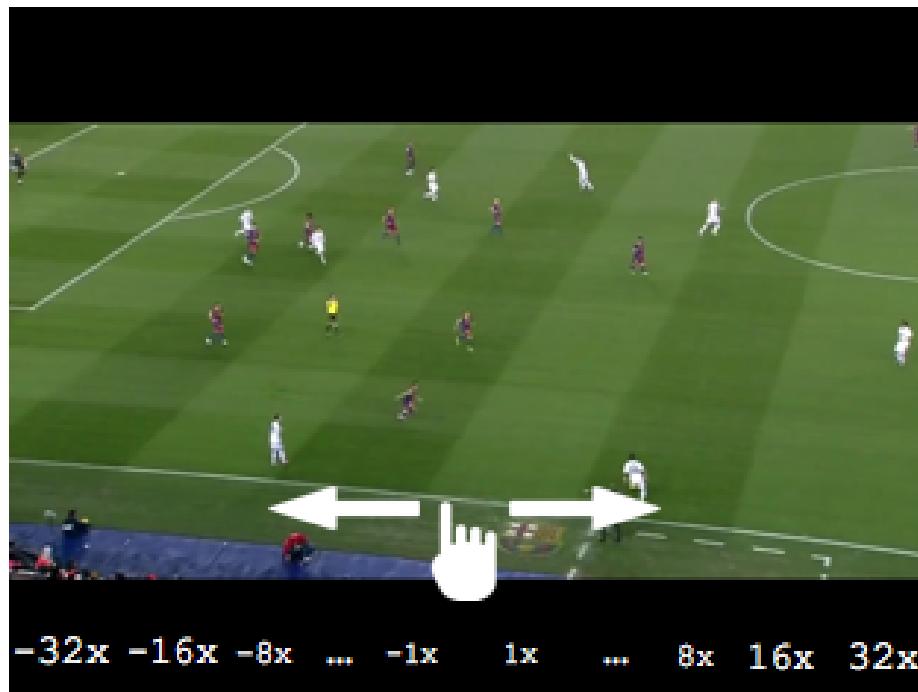


Abbildung 5.2: Flickplayer Wischgesteninteraktion

dann kann er das Video rückwärts abspielen. Sobald der Beginn erreicht wurde, springt man automatisch zum Ende.

5.1.3 Grafische Benutzeroberfläche

Die Benutzeroberfläche wurde so gestaltet, dass der Nutzer von den Elementen nicht abgelenkt wird und diese nur eine sekundäre Funktion als Unterstützung einnehmen. Grundsätzlich sollten auch nur die wichtigsten Elemente mit eingebracht werden. Der Nutzer muss zu jeder Zeit wissen können, an welcher Stelle er sich gerade befindet und in welchem Zustand der Navigation der Videoplayer ist, also wie schnell das Video gerade abgespielt wird. Das ist wichtig, da aufgrund des Inhaltes sich nicht immer erschließen lässt, wie schnell man das Video gerade abspielt. Insbesondere bei Videos mit wenig Bewegungen ist das der Fall. Abbildung 5.3 zeigt den Flickplayer auf einem Apple iPad mit einem Fußballvideo und den verschiedenen Elementen. Am unteren Rand des Bildschirms ist eine Zeitleiste, welche mit einem roten Balken immer den aktuellen Fortschritt anzeigt. Diese Zeitleiste lässt sich jedoch nicht zur Navigation verwenden, wie dies bei den Standardvideoplayern üblich ist. Am oberen Rand werden in jeder Ecke jeweils noch ein Textlabel angezeigt. Das in der rechten Ecke gibt die gesamte Videolänge in Minu-



Abbildung 5.3: Benutzeroberfläche des Flickplayers

ten und Sekunden wieder. Das Label in der linken oberen Ecke gibt immer die aktuelle Position wieder. Mittig am oberen Bildschirmrand ist ein Element, welches die aktuelle Wiedergabegeschwindigkeit anzeigt. Die Länge des Balkens deckt den gesamten Navigationsbereich ab. Ist der Balken bis zur Mitte hin gefüllt, bedeutet dies, dass der interne Wert bei 0 ist und die 1-fache Wiedergabe ist aktiv. Zusätzlich wird immer an der Stelle der aktuellen Füllmenge ein Indikator eingeblendet, welcher die Abspielgeschwindigkeit angibt. In der Abbildung liegt die momentane Geschwindigkeit bei 16-facher Rate. Mit diesen visuellen Elementen hat der Nutzer immer genauen Überblick über alle Parameter. Um das Nutzererlebnis zu verbessern wird bei einer schnelleren Abspielrate als der 1-fachen der Ton des Videos abgeschaltet, da dieser sehr unverständlich wird.

Wie die Anforderungen in der Implementierung umgesetzt wurden, wird im folgenden Kapitel detailliert dokumentiert.

5.2 Details zur Implementierung und Probleme

In diesem Kapitel wird die Implementierungsphase genauer beschrieben und die Probleme, die während der Umsetzung aufgetreten sind und wie diese gelöst wurden. Dabei wird auch die verwendete Hard- und Software, einzelne Klassen und Methoden bis hin zum Logging für die Benutzerstudie erläutert.

5.2.1 Hardware und Software

Die Zielplattform des Prototyps ist mit Apple iOS an bestimmte Hardware und Software gebunden. Da die Applikation einen erhöhten Ressourcenbedarf hat, war das Testgerät ein Apple iPad Air der ersten Generation. Zum Zeitpunkt der Implementierung war dies die aktuellste und leistungsstärkste iPad Version. Das Zielbetriebssystem war mit iOS 8 ebenfalls die aktuellste Softwareversion. Um Applikationen für iOS zu entwickeln, ist man als Programmierer an Apple's eigene Entwicklungsumgebung Xcode gebunden. Da die Umsetzung des Konzepts von Beginn an verschiedene Probleme zu bewältigen hatte, wurden verschiedene Ansätze konzipiert. Um diese unabhängig voneinander testen zu können, wurde das Versionierungssystem GIT verwendet. Dadurch konnte man mühelos zwischen den verschiedenen Versionen wechseln, ohne dass die unterschiedlichen Codefragmente Auswirkungen aufeinander haben. Des Weiteren wurde das Repository auf der Codehosting Plattform *Github* hochgeladen, sodass im Falle von Defekten auf dem Entwicklungsgerät kein Datenverlust auftreten kann und immer eine Kopie des Quellcodes vorhanden ist [67]. Eines der Probleme war die Kodierung der Videos, welches im Detail im Unterkapitel „Probleme“ weiter behandelt wird. Jedoch kann hier schon angemerkt werden, dass *FFmpeg* als Werkzeug verwendet wurde, um eines dieser Probleme zu lösen. FFmpeg ist ein plattform-übergreifendes, quell-offenes Projekt, welche eine Vielzahl von leistungsstarken Programmen und Softwarebibliotheken zusammenfasst, um Video und Audiodateien aufzunehmen, in verschiedene Formate umzuwandeln und zu manipulieren [68]. Das Tool an sich wird über die Kommandozeile bedient, wobei es mittlerweile auch Programme wie *ffmpegx* gibt, die eine Benutzeroberfläche haben und die wichtigsten Funktionen von FFmpeg so einfacher zu bedienen sind. [69]

5.2.2 Probleme

Eines der Hauptprobleme während der Implementierung, war die geringe Leistungsfähigkeit des iPads. Die Video-Engine des AVFoundation Frameworks ist dahingehend limitiert, dass ein Video nur bis maximal 2-facher Geschwindigkeit abgespielt werden kann, bevor das Video beim Abspielen ruckelt. Daher war der erste Ansatz, also auf Basis der Nutzerinteraktion die Abspielrate dynamisch anzupassen, so nicht mehr möglich.

Die Videos mussten daher zuvor dahingehend bearbeitet werden, dass diese in beschleunigter Form gespeichert wurden. Mit Hilfe von FFmpeg wurden die Videos mit 4-facher und 16-facher Geschwindigkeit kodiert, wodurch pro Video drei Videodateien benötigt wurden. Die Intervalle für die Abspielrate konnte so optimal abgedeckt werden, da beispielsweise das Video, welches in 16-facher Geschwindigkeit kodiert wurde, mit zweifacher Rate abgespielt werden konnte und so der Nutzer das zu betrachtende Video in 32-facher Geschwindigkeit wahrnimmt. Während der Wiedergabe musste dann zwischen den drei Dateien gewechselt werden, ohne dass dies dem Betrachter auffällt. Wie dies im Detail umgesetzt wurde, wird in den folgenden Kapiteln genauer erklärt. Ein weiteres Problem, welches es zu lösen galt, war die rückwärts-Wiedergabe. Ein Video besteht aus unterschiedlichen Arten von Einzelbildern, wie in Abbildung 5.4 zu sehen ist. Im Einzelnen sind das die I-Frames, auch Inter-Frames genannt, welche unabhängig von anderen Einzelbildern die gesamte Information innehaben. Des Weiteren gibt es noch die P-Frames, welche nur Informationen über die Unterschiede zum vorherigen Bild speichern. Die sogenannten B-Frames speichern sowohl die Unterschiede zum vorherigen und zum nächsten Bild, wodurch eine besonders hohe Komprimierung erreicht werden kann. Nun gibt es das Problem, wenn man ein ganz bestimmtes Frame anzeigen möchte und dieses kein I-Frame ist, denn dann muss das letzte I-Frame als Startpunkt genommen werden und alle Frames in der Kette bis hin zum Ziel-Frame berechnet werden. Alle Frames von einem I-Frame zum nächsten wird auch *Group of Pictures*, kurz GoP, genannt, was auf der Abbildung 5.4 zu sehen ist [23]. Da die Berechnung der Zwischenframes einige Zeit

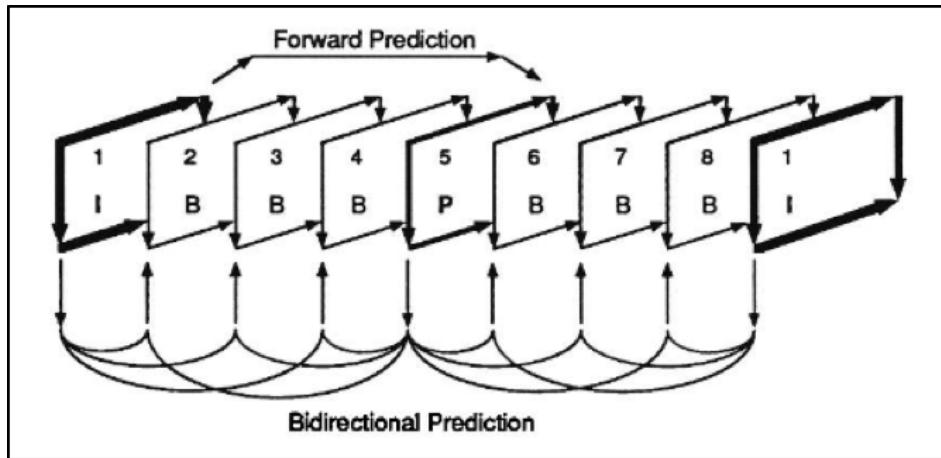


Abbildung 5.4: Frame-Typen der Videokodierung [23]

dauert, kann das Video nicht ruckelfrei rückwärts abgespielt werden [70]. Die Lösung hierfür war alle Videos zusätzlich so zu kodieren, dass sie nur aus I-Frames bestehen,

sprich die GoP Größe wird auf 1 gesetzt. Dadurch war nur eine Einzelbildkomprimierung möglich, da jedes Frame ein Vollbild ist. Für das Evaluieren der Navigationsmethode war dies jedoch vollkommen ausreichend.

5.2.3 Storyboard und Klassen

In diesem Kapitel wird auf die Projektarchitektur eingegangen, also wie die Klassen und View Controller im Storyboard strukturiert sind, sowie kurze Erläuterungen zu wichtigen Methoden. Der Prototyp sollte sich nur auf die Anforderungen, welche in Kapitel 5.1 beschrieben wurden, fokussieren. Aus diesem Grund konnte die Projektgröße klein gehalten werden. Das gesamte Storyboard besteht, hier in Abbildung 5.5 zu sehen, aus zwei Einheiten. Zum einen ist das der Navigation Controller, welcher im Navigationstack die Übergänge und Navigation zwischen den verschiedenen View Controllern handhabt. An zweiter Position ist die Collection View zu sehen. Dieser gibt eine Übersicht über alle Videos, die von iTunes auf das iPad kopiert wurden, sodass diese zugänglich sind. Diese Collection View enthält Zellen, eine für jedes verfügbare Video, jeweils mit einem Vorschaubild und dem Dateinamen. Das wichtigste Element im Storyboard ist der dritte View Controller der Abbildung. Dieser implementiert die Wiedergabe und Navigation des Flickplayers, also die Kernaufgabe der Applikation. Ein vierter View Controller ist ebenfalls auf dem Storyboard zu finden, dieser ist aber nur ein Eingabeformular für die Teilnehmerdaten der Benutzerstudie und hat keinen Einfluss auf die eigentlichen Navigation. Zu beiden View Controllern gibt es jeweils eine Klasse im Projekt, sowie eine



Abbildung 5.5: Storyboard des Flickplayers

Klasse für die Zelle im Collection View. Alle Klassennamen haben den Präfix VP, was eine Abkürzung für Videoplayer ist. Die wichtigste Methode in der Klasse VPCollectionViewController ist dazu da, um alle Videos aus dem Dateisystem zu lesen.

```
//get the iPod library
- (void) buildVideoLibrary {
    itemList = [[NSMutableArray alloc] init];
```

```
MPMediaPropertyPredicate *predicate = [MPMediaPropertyPredicate
    predicateWithValue:[NSNumber numberWithInteger:
        MPMediaTypeHomeVideo] forProperty:
        MPMediaItemPropertyMediaType];
MPMediaQuery *query = [[MPMediaQuery alloc] init];
[query addFilterPredicate:predicate];
[itemList addObjectFromArray:[query items]];
}
```

Das Attribut `itemList` wird hier zuerst initialisiert, bevor die Medienobjekte zugewiesen werden können. Um nicht alle Mediendateien vom System zu erhalten, muss ein Filter, auch `predicate` genannt, erstellt werden. Dazu wird ein Objekt der Klasse `MPMediaPropertyPredicate` initialisiert. Der Konstruktor verlangt dafür einen Filterwert, welcher aus der `MPMediaType` Enumeration stammt. Um Videos zu erhalten, die manuell über iTunes auf das Gerät kopiert wurden, muss der Wert `MPMediaTypeHomeVideo` übergeben werden. Anschließend wird dieser Filter einer `MPMediaQuery` Instanz übergeben. Dessen Attribut `items` hält nun alle Videos, die gesucht wurden [71]. Mit der Klasse `AVAssetImageGenerator` ist es nun möglich ein Vorschaubild der Videos zu erzeugen, die dann in den Zellen angezeigt werden können. Von der Collection View Zelle gibt es einen Segue, also eine Art Verknüpfung innerhalb des Storyboard zu einem anderen View Controller, welcher den eigentlich Video View Controller öffnet, um das jeweilige Video abzuspielen [72]. Beim Initialisieren des `VPPPlaybackViewController` werden noch nötige Attribute gesetzt. Diese sind die drei Video Dateien, in den verschiedenen Abspielgeschwindigkeiten 1-fach/4-fach/16-fach, die zum Abspielen eines Videos benötigt werden. In Kapitel 2.4.4 wurde erläutert, wie ein Videoplayer initialisiert und dem Hauptview des View Controllers zugewiesen werden muss. Beim Flickplayer übernimmt diese Aufgabe die Methode `-(void)setupPlayers;`, welche in der `- (void)viewDidLoad;` Methode aufgerufen wird. Jeder View Controller bietet einige Methoden, die ausgeführt werden, wenn der View Controller beispielsweise geladen, in der Vordergrund kommt oder auch wieder geschlossen wird. Die `viewDidLoad` Methode wird immer dann ausgeführt, wenn der View Controller geladen wird, also noch bevor dieser angezeigt wird. In der Regel werden innerhalb dieser Methode Vorbereitungen gemacht, wie Anpassen der Layout Elemente, oder wie am Beispiel des Flickplayers, das Laden der Videodateien und übergeben an die `AVPlayer` Instanzen [73]. Neben kleineren Benutzerelementen werden hier auch die beiden Timer initialisiert, die für die Synchronisierung der drei Videoplayer zuständig sind, sowie automatische Anpassung der Abspielgeschwindigkeit. Wie dies im Detail umgesetzt wurde, wird im folgenden Kapitel genauer erläutert. Teil der Implementierung dieser Klasse, waren auch drei der Methoden aus der `UIResponder` Klasse. Diese sind:

- - (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
- - (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
- - (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event

Die touchesBegan Methode wird ausgeführt, sobald der Nutzer mit dem Finger den Bildschirm berührt. Das UIEvent, welcher der Methode als Parameter übergeben wird, enthält alle Informationen, die benötigt werden. Das sind unter anderem die exakte Position auf dem Bildschirm, sowie der Zeitstempel, also wann die Berührung gestartet wurde. Wie der Name der Methode schon vermuten lässt, wird diese Methode pro Interaktion genau einmal ausgeführt. Solange sich der oder die Finger auf dem Bildschirm bewegen, wird die Methode touchesMoved aufgerufen und zum Abschluss auch noch die Methode touchesEnded, wenn die Interaktion durch das Abheben des Fingers beendet wird. Welche Schritte innerhalb dieser Methoden ablaufen, wird ebenso im folgenden Kapitel genau ausgearbeitet. Der Videoplayer bietet wie in Kapitel 5.1.3 Elemente, die dem Nutzer dabei helfen eine Übersicht über die Position und Navigation zu erhalten. Diese Elemente werden ebenfalls in dieser Klasse geladen und aktualisiert. Die folgende Methode updateTimeLabel wird 12-mal pro Sekunde vom selben Timer aufgerufen, welcher die drei Videoplayer synchronisiert.

```
-(void)updateTimeLabel {
    self.progressBar.progress = CMTimeGetSeconds(player1.currentTime) /
        CMTimeGetSeconds(playerItem1.duration);
    double seconds = CMTimeGetSeconds([player1 currentTime]);
    if (!isfinite(seconds)) {
        seconds = 0;
    }
    int secondsInt = round(seconds);
    int minutes = secondsInt / 60;
    secondsInt -= minutes * 60;
    int secondsIntDuration =
        round(CMTimeGetSeconds(player1.currentItem.duration));
    int minutesDuration = secondsIntDuration / 60;
    secondsIntDuration -= minutesDuration * 60;
    self.currentTimeLabel.text = [NSString stringWithFormat:@"%.2i:%.2i",
        minutes, secondsInt];
    self.totalLengthLabel.text = [NSString stringWithFormat:@"%.2i:%.2i",
        minutesDuration, secondsIntDuration];
}
```

Der `UIProgressView` `progressBar` wird zuerst auf den neusten Stand gebracht. Dabei wird vom aktuell abgespielten Video die momentane Position genommen und durch die gesamte Videodauer geteilt. Daraus ergibt sich ein Prozentwert, welcher dann der `progressBar` als Wert zur Füllung übergeben wird. Des Weiteren werden in dieser Funktion die beiden Textlabels aktualisiert, die jeweils in die oberen Ecken positioniert sind. Dazu werden sowohl die gesamte Videodauer, sowie die momentane Position als Text umgewandelt, sodass diese Werte jeweils auf den Labels im `mm:ss` Format angezeigt werden können.

Für die Statusanzeige der aktuellen Abspielgeschwindigkeit wurde, um die Komplexität einzugrenzen, auf ein Open Source Projekt zurückgegriffen. Der `ASProgressPopUpView` auf Github ist eine Subklasse der Standard-Progress Bar von iOS. Zusätzlich zum Auffüllen der Leiste, gibt es noch die Funktion ein Textlabel direkt an der Füllposition anzuhafeten, welches dynamisch den Text, sowie die Farbe anpassen kann. Die Farbanpassung, je nach Abspielgeschwindigkeit, sollte dem Nutzer als zusätzliche visuelle Hilfe die Navigation vereinfachen. Die Füllfarben wurden so gewählt, dass ein flüssiger



Abbildung 5.6: Geschwindigkeitsindikator Flickplayer

Übergang von Rot nach Blau erfolgt, wie auch in der Abbildung 5.6 zu sehen ist.

```
-(void)setupSpeedBar {
    self.progressViewMinus.popUpViewAnimatedColors = @[[UIColor redColor], [UIColor blueColor]];
    self.progressViewMinus.popUpViewCornerRadius = 10.0;
}
```

Zwischen der maximalen negativen und der maximalen positiven Geschwindigkeit gibt es insgesamt 12 Stufen. Der Timer, welcher die Geschwindigkeitsanpassung während der Navigation implementiert, wird 20-mal pro Sekunde aufgerufen. Die `ASProgressPopUpView` Klasse verfügt über ein Protokoll, welches vom View Controller implementiert werden muss, um den Text zu aktualisieren. Sobald der Füllstatus sich verändert, wird die folgende Protokollmethode ausgeführt:

```
- (NSString *)progressView:(ASProgressPopUpView *)progressView
    stringForProgress:(float)progress
```

Als Überabeparameter erhält der Methodenblock den Prozentwert der Füllung. Mit

einer einfachen Berechnung aller Intervalle $x/12$ wird dann überprüft, in welchem Intervall sich dieser übergebene Werte befindet. Entsprechend wird dann der Text der Labels gesetzt. [74] Im folgenden Kapitel wird im Detail aufgeführt, wie die Navigation implementiert wurde.

5.2.4 Umsetzung des Interaktionskonzepts

Die Implementierung der Interaktionsmethode hatte einige Hürden, auf die schon in Kapitel 5.2.2 eingegangen wurde. Für die Umsetzung wurden verschiedene Methoden geschrieben, die beispielsweise die Videos synchronisieren, nach der Beschleunigung automatisch wieder die Geschwindigkeit auf 1-fache Abspielrate reduzieren, die drei verschiedenen Views der einzelnen Videos in den Vordergrund bzw. Hintergrund zu schieben, sowie die Implementierungen der `UIResponder`-Methoden. Die Details dieser Methoden werden in diesem Kapitel verarbeitet.

Die Lösung für das Problem mit hohen Abspielraten war, dass die Videos vorkodiert auf das Gerät übertragen wurden und dann übereinandergelegt wurden, sodass zwischen den einzelnen Videos hin und her geschalten werden konnte. Jede `UIView`, welche jeweils ein Video hält, wurde so initialisiert, dass der Bildschirm vollständig ausgefüllt wird. In Abbildung 5.7 ist dargestellt wie die drei `UIViews` auf dem Bildschirm liegen, und welche View, welches Video mit der jeweiligen Geschwindigkeit darstellt. Die `UIResponder` Events werden immer von der ersten `UIView` entgegengenommen, da diese Methoden für diese View aber nicht implementiert wurden, werden diese durch die Stack Hierarchie weitergegeben, bis hin zum `VPPPlaybackViewController`, welcher eine Unterklasse von `UIViewController` ist und somit auch ein `UIResponder`. Der `VPPPlaybackViewController` hält ein Gleitkomma Attribut `speed`, welches der zentrale Wert für die aktuelle Abspielgeschwindigkeit ist. In der `touchesBegan` Methode werden die Werte über die Position der Berührungen auf der X-Achse gespeichert, sowie dessen Zeitpunkt. Die Y-Position kann hier bewusst vernachlässigt werden, da vertikale Wischgesten nicht verwendet werden. Wenn sich der Finger über den Bildschirm bewegt, dann wird bei jeder kleinen Zwischenbewegung die Methode `touchesMoved` aufgerufen.

```
-(void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
    if (player1.rate != 0.0 || player2.rate != 0.0 || player3.rate != 0.0) {
        UITouch *touch = [[event allTouches] anyObject];
        CGPoint touchLocation = [touch locationInView:self.view];
        self.speed = self.speed - 3.3 * (touchLocation.x - self.position) / (event.timestamp - self.time);
        if (self.speed < -180000.0) {
            self.speed = -180000.0;
        } else if (self.speed > 180000.0) {
```



Abbildung 5.7: Player View Stack

```
        self.speed = 180000.0;
    }
    self.time = event.timestamp;
    self.position = touchLocation.x;
}
}
```

Der Videoplayer bietet auch eine Funktion, um die Videos zu pausieren. Aus diesem Grund wird zuerst überprüft, ob eines der Videos abgespielt wird. Falls nicht, dann wir diese Methode keine Auswirkungen haben. Ansonsten wird auch hier die X-Achsen Position gespeichert, um bei einem Vergleich zum vorherigen Wert aus der touchesBegan Methode festzustellen, in welche Richtung horizontal gewischt wurde. Die Bewegungsdistanz wird in Relation zum zeitlichen Abstand der Ereignisse gesetzt und das Ergebnis mit 3,3 multipliziert. Das Ergebnis aus dieser Gleichung wird zum aktuellen speed Wert addiert. Je nach Bewegungsrichtung kann das Ergebnis sowohl positiv, als auch negativ ausfallen. Der letzte Wert der X-Achsen Position, sowie der Zeitstempel werden immer gespeichert, um als Referenz verwendet zu werden, wenn die Bewegung weiter ausgeführt wird und so die touchesMoved Methode wieder aufgerufen wird. Um eine obere und

untere Grenze zu schaffen, kann `speed` nur Werte annehmen, die zwischen -180.000 und 180.000 liegen. Der Multiplikator wurde nicht zufällig gewählt, sondern wurde in kurzen Zwischentest mit Probanden so lange angenähert, bis eine positive Rückmeldung von den Nutzern kam, dass diese Konfiguration am angenehmsten zu verwenden ist. Denn umso höher dieser Multiplikator ist, desto einfacher ist es möglich auf die maximale Abspielrate von 3200% zu gelangen.

Damit das Attribut `speed` auch ohne Benutzereingabe von alleine wieder in den Ausgangszustand gelangt, gibt es eine Easing-Funktion, die diesen linear reduziert. Hier wurden vom einem Extrem zum Nullpunkt insgesamt 12 gleich große Intervalle definiert. Innerhalb einer Sekunde wird der Wert 20-mal um 750 reduziert, sodass nach maximal 12 Sekunden wieder der Normalzustand angenommen und das Video in einfacher Rate abgespielt wird. Bei jedem Aufruf der Easing-Funktion, wird nach Ausführung berechnet, welcher Videoplayer aktiv sein sollte und mit welcher Geschwindigkeit dieser seinen Inhalt wiedergeben soll. Dabei wird das Attribut `activePlayer` gesetzt, welches einen Ganzzahlwert zwischen 1 und 6 annehmen kann. Dies entspricht den drei Videos, die mit 100% und 200% ihrer Abspielrate jeweils zwei Zustände annehmen können. Auf das Klassenattribut `activePlayer` wurde ein Key-Value Observer gesetzt, welche schon in Kapitel 2.4.3 genauer erläutert wurden. Sobald also sich der Wert des Attributs ändert, wird folgende Methode ausgeführt:

```
-(void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object  
change:(NSDictionary *)change context:(void *)context;
```

Der `keyPath`, welcher als Parameter mitgegeben wird, entspricht dann dem Attributnamen, auf welchem der Observer gesetzt wurde. In dieser Methode werden alle Informationen verarbeitet, um das Video abzuspielen, welches im Vordergrund sein sollte, sowie die dessen Abspielrate festgelegt. Zuerst wird in einer if-Anweisung geprüft, welcher Player der aktuelle Player sein muss. Beispielsweise kann das momentane aktive Intervall, also der `activePlayer`, gleich zwei sein. Das bedeutet, dass das Video, welches in 1-facher Geschwindigkeit kodiert ist, in der Vordergrund kommt und die beiden anderen Videos pausiert werden. Des Weiteren gibt dieses Intervall an, dass die Rate auf 200% gesetzt werden muss. Das Ergebnis ist das Video, welches in doppelter Geschwindigkeit dem Nutzer wiedergegeben wird. Falls das Attribut `activePlayer` den Wert 4 angenommen hat, dann wird das 4-fach Video mit zweifacher Geschwindigkeit und somit 8-fach dem Nutzer präsentiert. Die exakten Intervalle mit den verschiedenen Abspielgeschwindigkeiten, sind in Abbildung 5.8 grafisch dargestellt. Auf der linken Seite sind alle möglichen positiven Abspielrate aufgelistet. Ebenfalls zu sehen sind alle Intervalle von 0 bis 180.000, welche Werte vom Attribut `speed` sind. Auf der linken Seite sind die drei Videos mit ihren kodierten Geschwindigkeiten markiert. Die Diagonale ist die Easing-

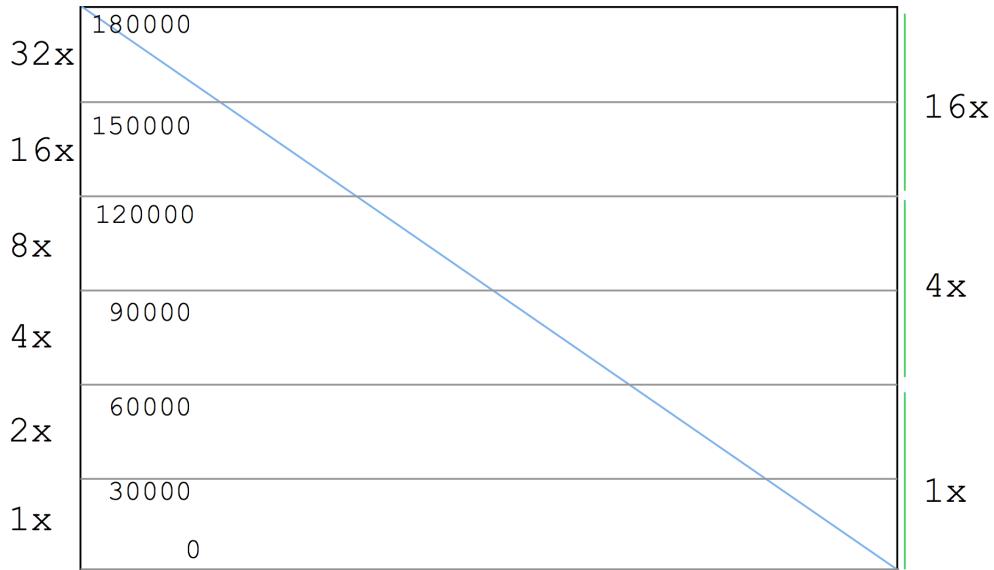


Abbildung 5.8: Flickplayer Geschwindigkeits-Intervalle

Funktion, welche die Geschwindigkeit reduziert, falls der Nutzer keine Eingabe macht. Die gesamte Dauer der Linie vom Beginn, bis zum Nullpunkt entspricht 12 Sekunden. Diese Zeit wurde von Testern während der Implementierungsphase als angenehm empfunden, da jede Zwischengeschwindigkeit zwei Sekunden lang dargestellt wird. Da das Video auch rückwärts abgespielt werden kann, ist das implementierte Prinzip vollständig in den negativen Bereich ausgebrettet, sodass die Grafik bis auf -180.000 ausgeweitet werden kann.

Das Foundation Framework bietet mit `NSTimer` eine Klasse, die über Instanz Methoden verfügt, um bestimmte Aktionen in einem zeitlichen Abstand einmal oder auch regelmäßig auszuführen [75]. Beim Flickplayer wurde dies unter anderem eingesetzt, um die drei Videos miteinander synchron zu halten. Der `NSTimer` wurde dabei so initialisiert, dass die Funktion `adaptTimeStamps` 12-mal in der Sekunde ausgeführt.

```
defaultSpeedActiveTimer = [NSTimer scheduledTimerWithTimeInterval:1.5
    target:self selector:@selector(defaultSpeedActive) userInfo:nil
    repeats:YES];
```

Da der Flickplayer immer nur das Video abspielt, welches gerade im Vordergrund erscheint, sind die anderen beiden pausiert. Um nun festzustellen, welche Videos in ihrer Position aktualisiert werden müssen, ist es notwendig herauszufinden, welches Video gerade abgespielt wird.

```

- (void)adaptTimeStamps {
    CMTime videoPosition;
    Float64 videoPositionInSeconds;
    CMTimeScale scale = player1.currentItem.asset.duration.timescale;

    if (player1.rate != 0.0) {
        videoPosition = player1.currentTime;
        videoPositionInSeconds = (Float64) videoPosition.value /
            videoPosition.timescale;
        CMTime time1 = CMTimeMakeWithSeconds(videoPositionInSeconds
            / 4.0, scale);
        CMTime time2 = CMTimeMakeWithSeconds(videoPositionInSeconds
            / 16.0, scale);
        [player2 seekToTime: time1 toleranceBefore: kCMTimeZero
            toleranceAfter: kCMTimeZero];
        [player3 seekToTime: time2 toleranceBefore: kCMTimeZero
            toleranceAfter: kCMTimeZero];
    } else if (player2.rate != 0.0) {
        videoPosition = player2.currentTime;
        videoPositionInSeconds = (Float64) videoPosition.value /
            videoPosition.timescale;
        CMTime time1 = CMTimeMakeWithSeconds(videoPositionInSeconds
            * 4.0, scale);
        CMTime time2 = CMTimeMakeWithSeconds(videoPositionInSeconds
            / 4.0, scale);
        [player1 seekToTime: time1 toleranceBefore: kCMTimeZero
            toleranceAfter: kCMTimeZero];
        [player3 seekToTime: time2 toleranceBefore: kCMTimeZero
            toleranceAfter: kCMTimeZero];
    } else if (player3.rate != 0.0) {
        videoPosition = player3.currentTime;
        videoPositionInSeconds = (Float64) videoPosition.value /
            videoPosition.timescale;
        CMTime time1 = CMTimeMakeWithSeconds(videoPositionInSeconds
            * 16.0, scale);
        CMTime time2 = CMTimeMakeWithSeconds(videoPositionInSeconds
            * 4.0, scale);
        [player1 seekToTime: time1 toleranceBefore: kCMTimeZero
            toleranceAfter: kCMTimeZero];
        [player2 seekToTime: time2 toleranceBefore: kCMTimeZero
            toleranceAfter: kCMTimeZero];
    }
    [self updateTimeLabel];
}

```

Mit einer if-Anweisung wird überprüft, welches Video nicht pausiert, also die Rate des

AVPlayers nicht 0 ist. Die aktuelle Zeit des aktiven Videos wird erhalten und deren Basis und mit dem Umrechnungsfaktor die neue Position für die anderen beiden Videos errechnet. Beispielsweise wenn das erste Video gerade abgespielt wird, also das Video mit 1-facher Geschwindigkeit und an Stelle 16:00 ist, dann muss die aktuelle Zeit durch 4, bzw. 16 geteilt werden, um jeweils für die Videos mit 4- und 16-facher Geschwindigkeit den selben Inhalt wiederzugeben. Die Synchronisierung ist hauptsächlich dafür wichtig, dass beim Umschalten von einer zur anderen View ein flüssiger Übergang entsteht und der Nutzer dies nicht sehen kann.

Für die Benutzerstudie war es notwendig einen Logging-Mechanismus zu implementieren, sodass alle Nutzereingaben aufgezeichnet werden konnten. Somit konnte sichergestellt werden, dass zusätzlich zum Fragebogen, welcher die subjektiven Eindrücke aufzeichnen sollte, auch quantitative Aussagen über die Leistungsfähigkeit des Flickplayers getroffen werden konnten.

5.2.5 Logging

Um die Daten später einfacher analysieren zu können, hat man sich dazu entschieden, die Daten in eine *CSV* Datei zu schreiben. CSV steht für *Comma-separated values* und bedeutet, dass alle Werte mit einem Komma oder einem Semikolon getrennt werden. Des Weiteren werden die verschiedenen Datensätze jeweils mit einer neuen Zeile von einander getrennt. Der Vorteil dabei ist, dass die Daten in den üblichen Tabellenkalkulationsprogrammen geöffnet werden können [76]. Um die Daten besser durchsuchen zu können, wurden die Log-Dateien in eine *MySQL*-Datenbank importiert, sodass mit *SQL*-Abfragen schneller gesucht werden konnte. Generell sollten so viele Information wie möglich während der Benutzerstudie gesammelt werden.

Bevor der Nutzer mit den Aufgaben beginnen konnte, mussten einige Informationen zu dem jeweiligen Testlauf in ein Formular eingegeben werden, sodass die Daten jedem Nutzer, sowie der jeweiligen Aufgabe, zugewiesen werden konnte. In Abbildung 5.9 ist eine Bildschirmaufnahme des Eingabeformulars zu sehen. Jeder Testkandidat hat eine eindeutige ID erhalten, damit die ausgefüllten Fragebögen ebenfalls zugeordnet werden konnten. Des Weiteren wurde ein Schalter für die Auswahl der Aufgabenart angebracht. Dieser war notwendig, da sich die Benutzeroberfläche abhängig von der Aufgabe leicht unterschieden hatte. Um die Daten nicht alle in eine Datei zu speichern, konnte auf dem Eingabeformular ebenfalls ausgewählt werden, in welche der beiden Dateien die Testinformation gespeichert werden. Eine dritte Datei wurde ebenfalls angelegt, da jeder Nutzer vor Beginn der eigentlichen Aufgaben die Videoplayer schon benutzen konnte, um sich mit den verschiedenen Navigationsmethoden vertraut zu machen. Die Informationen aus dem Eingabeformular konnten nun gespeichert werden und das Video, somit auch das

Logging, gestartet werden. Das Video wird immer sofort gestartet, sobald es ausgewählt

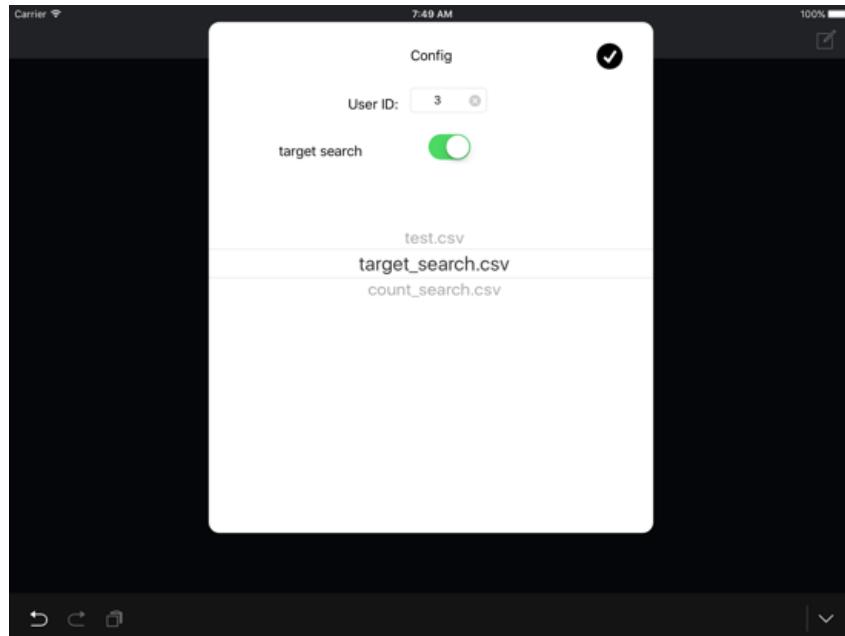


Abbildung 5.9: Formular für Nutzertestinformationen

wurde. Der erste Logeintrag wird genau zu diesem Zeitpunkt erzeugt. Jeder Datensatz hat eine Vielzahl von Spalten, die hier im Einzelnen erläutert werden.

- ID – Jeder Datensatz hat eine eindeutige ID.
- userID – Das ist die eindeutige Benutzer ID.
- filename – Der Dateinamen, um danach die Daten der Aufgabe zuordnen zu können.
- timestamp – Der genaue Zeitpunkt, an dem der Datensatz erzeugt wurde.
- currenttime – Die aktuelle Position innerhalb des Videos.
- playbackspeed – Die aktuelle Wiedergabegeschwindigkeit.
- isplaying – Ist das aktuelle Video pausiert oder abgespielt wird.
- pressed_button – Ein Boolean, ob dieser Datensatz durch das Benutzen der Zähler ausgelöst wurde.

- stepper_1, stepper_2, stepper_3, stepper_4 – Die Suchaufgaben für das Zählen von bestimmten Ereignissen boten einen Zähler für jedes Ereignis, welche auch in Abbildung 5.1 zu sehen sind. Diese Werte geben den aktuellen Zählstand wieder.
- touch_x - Die Fingerposition auf der horizontalen Ebene.
- touch_y – Die Fingerposition auf der vertikalen Ebene.
- acceleration_speed – Das ist der interne speed Wert.
- finger_touches_screen – Ein Boolean, ob der Nutzer den Bildschirm gerade berührt.
- start_end – Der Datensatz ist ein Start-Datensatz, oder der Datensatz beim Beenden.

Die Aufgaben mit dem Standardplayer hatten noch eine Besonderheit, denn hier wurde die Granularität beim Benutzen der Navigationsleiste mitgespeichert. Die Datensätze wurden an verschiedenen Zeitpunkten erstellt. Der erste Eintrag wird erstellt, sobald das Video gestartet wird. Des Weiteren werden jeweils Logeinträge in den Methoden touchesBegan, touchesMoved und touchesEnded gespeichert. Bei den Aufgaben, die Zähler benötigen, wird jeweils noch ein Eintrag geloggt, sobald einer der vier Zähler betätigt wurde. Beim Beenden der Aufgabe wird noch ein letzter Logeintrag gespeichert, um das Ende des Tests zu kennzeichnen. Die Aufgaben, die mit dem Standardplayer durchgeführt wurden, erstellen Logeinträge, wenn der Nutzer die Navigationsleiste bediente, da hier die UIResponder Methoden nicht verwendet werden.

Der Aufbau der Studie, sowie deren Ausführung und Analyse der gewonnenen Daten, werden im folgenden Kapitel genauer erörtert.

6 Benutzerstudie und Evaluierung

Um die Performanz des vorgeschlagenen Flickplayers zu messen, wurde im Rahmen der Arbeit eine Benutzerstudie durchgeführt. Dabei wurden den Probanden verschiedene Suchaufgaben gestellt, die sie sowohl mit dem Prototyp, als auch mit dem Standardplayer erledigen mussten. Durch den direkten Vergleich sollte sichergestellt werden, dass die Unterschiede zueinander Rückschlüsse auf die Effektivität und die Nutzerakzeptanz zulassen. Um so viele Daten wie möglich zu sammeln, wurde bei der Implementierung der Applikationen eine Logging-Funktion umgesetzt, welche alle Interaktionen der Benutzer mit den Videoplayern aufzeichnet. Zur Ergänzung wurde mit Hilfe eines zweiteiligen Fragebogens das subjektive Empfinden der Probanden festgehalten. Die Details des Versuchsaufbaus, die Durchführung und Analyse der gesammelten Daten werden in diesem Kapitel behandelt. Im Anschluss werden die gewonnenen Ergebnisse interpretiert und präsentiert.

6.1 Aufbau der Studie

Für die Durchführung der Studie wurden insgesamt 16 Probanden befragt. Jeder Benutzer musste sowohl Aufgaben mit dem Standardvideoplayer erledigen, als auch mit dem Prototyp des Flickplayers. Auf beiden Applikationen waren die selben neun Videos verfügbar, wobei eines davon ein etwa 15 Minuten langes Testvideo war, mit dem die Nutzer sich mit den jeweiligen Navigationsmethoden vertraut machen konnten, bevor die eigentlichen Aufgaben durchgeführt wurden. Die Daten, die während des Testvideos aufgezeichnet wurden, sind aus der Analyse herausgelassen worden. Die Aufgaben lassen sich in zwei Kategorien einteilen. Zum einen sind die Aufgaben aus dem Bereich Zielsuche, also bei dem der Nutzer vor Beginn eine bestimmte Sequenz mit einer Länge von 20 Sekunden gezeigt bekommt und anschließend diese Szene in einem etwa 45 Minuten langem Video so schnell wie möglich finden muss. Dabei weiß der Nutzer nicht an

welcher Stelle sich diese Szene befindet. Und zum anderen Suchaufgaben, bei denen in einem 15-minütigen Video vier verschiedene Ereignisse gezählt werden müssen, die beliebig oft vorkommen können. Der Grund für diese Auswahl der Aufgaben war, dass somit zum einen die Schnelligkeit untersucht werden kann, wie lange ein Nutzer braucht, um ihm bekannten Szenen ausfindig zu machen und zum anderen wie viel von den Inhalten der Videos aufgenommen wird, wenn der Nutzer mit teilweise stark erhöhter Abspielgeschwindigkeit durch die Videos navigiert. Für die Suchaufgaben, bei denen die Nutzer



Abbildung 6.1: Prototypen mit Adaptionen für Studie

vorgegebene Ereignisse finden und zählen müssen, wurden beide Videoplayer Applikationen um ein Widget ergänzt, mit denen das Zählen vorgenommen werden musste. Dieser ist in Abbildung 6.1 zu sehen. Das betätigen der Zähl-Knöpfe wurden ebenfalls durch das Logging-System mitgeschrieben, sodass diese Daten einfacher und genauer ausgewertet werden konnten. Wenn der Nutzer die Aufgabe beenden möchte, dann ist dafür in der unteren rechten Ecke ein roter Knopf. Die Anzahl der Testpersonen wurde bewusst so gewählt, dass alle Permutationen der Aufgaben abgedeckt werden. In der folgenden Abbildung 6.2 ist der exakte Versuchsaufbau detailliert dargestellt. Insgesamt wurden acht verschiedene Videos in der Studie verwendet. Diese wurden in vier verschiedene Arten unterteilt, die da wären Fußball, Quizshow, Dokumentation und Nachrichten. Von jeder Art gab es jeweils zwei verschiedene Videos. Welches Video für welchen Videoplayer und welchen Testkandidaten verwendet wurde, kann in der Abbildung 6.2 bei der Unterscheidung von 1 und 2 gesehen werden. Von Benutzer zu Benutzer wurde immer abgewechselt, mit welchem Videoplayer die Aufgaben zuerst bearbeitet werden mussten. Dies führte dazu, dass der erste Nutzer mit dem sogenannten Flickplayer und der zweite Nutzer mit dem Standardplaner startete. Des Weiteren wurde ebenso zuvor definiert, mit welcher Art von Aufgaben begonnen werden musste. In der ersten Spalte kann dies

Beutzer	Beginnt mit	event counting		known-item-search		event counting		known-item-search	
		Fußball	Millionenshow	Dokumentation	Nachrichten	Fußball	Millionenshow	Dokumentation	Nachrichten
1	Flick	1	1	1	1	2	2	2	2
2	Standard	1	1	1	2	2	2	2	1
3	Flick	1	1	2	1	2	2	1	2
4	Standard	1	1	2	2	2	2	1	1
5	Flick	1	2	1	1	2	1	2	2
6	Standard	1	2	1	2	2	1	2	1
7	Flick	1	2	2	1	2	1	1	2
8	Standard	1	2	2	2	2	1	1	1
9	Flick	2	1	1	1	1	2	2	2
10	Standard	2	1	1	2	1	2	2	1
11	Flick	2	1	2	1	1	2	1	2
12	Standard	2	1	2	2	1	2	1	1
13	Flick	2	2	1	1	1	1	2	2
14	Standard	2	2	1	2	1	1	2	1
15	Flick	2	2	2	1	1	1	1	2
16	Standard	2	2	2	2	1	1	1	1

Abbildung 6.2: Aufbau der Benutzerstudie

zwischen den Farben Gelb und Grün erkannt werden. Gelb stand in dieser Studie für die Zielsuchaufgaben und Grün für die Aufgaben, bei denen die Probanden bestimmte Ereignisse zählen mussten. Damit die Testkandidaten ihr angeeignetes Wissen über die Videos nicht bei anderen Aufgaben anwenden konnten, wurde bei der Verteilung darauf geachtet, dass kein Kandidat ein Video mehrfach bearbeitet. Die Permutationen ergaben somit eine Anzahl von 16 Konfigurationen, die getestet werden mussten, um eine aussagekräftige Studie zu gewährleisten.

6.2 Durchführung

Für die Durchführung wurde ein schwarzes Apple iPad Air 1 mit einem 9,7 Zoll großen Bildschirm verwendet. Dabei wurden bei beiden Applikationen die Videos im Querbildformat abgespielt, ohne den Nutzern die Möglichkeit zu bieten, die Ausrichtung in ein Hochbildformat zu wechseln. Der Grund dafür war, dass alle Videos im Querformat kodiert waren. Von den Teilnehmern waren insgesamt sechs Frauen und zehn Männer im Alter von 18 bis 31 Jahren. Das Durchschnittsalter lag bei 25,38 Jahren mit einer Standardabweichung von $\pm 3,2$. Die Voraussetzung um bei der Studie teilzunehmen war, dass man mindestens ein Jahr Erfahrung mit dem Umgang von Smartphones oder Tablets hat. Von den 16 Teilnehmern hat keiner weniger als drei Jahre und zwei der Teilnehmer sogar acht Jahre Erfahrung. Im Durchschnitt benutzen die Probanden seit 5,125 Jahren Smartphones und Tablets, mit einer Standardabweichung von $\pm 1,54$. Mit einer leichten Mehrheit von 56% waren die Teilnehmer zudem Brillen, bzw. Kontaktlinsenträger. Um eine gleichbleibende Umgebung zu schaffen, wurde die Studie für alle Probanden immer

im selben Raum durchgeführt, welcher bis auf einen Tisch und zwei Stühle vollständig leer war. Des Weiteren wurden äußere Einflüsse vermieden, indem immer nur ein Proband, sowie der Untersuchungsteilnehmer gleichzeitig im Raum waren. Um die Konfiguration aus Abbildung 6.2 verständlicher zu machen, wird im Folgenden die Aufgabenverteilung des ersten Benutzers erläutert. In der ersten Spalte der Tabelle wird anhand der gelben Ausfärbung erkannt, dass Benutzer 1 mit den Zielsuchaufgaben beginnt. In der zweiten Spalte gibt es zudem noch die Information, dass der Prototyp des Flickplayers als erstes verwendet werden muss. Der blaue Block steht hierbei für die Aufgaben, die mit dem Flickplayer erledigt werden müssen und der orangefarbene Block für die des Standardplayers. Bevor der Testkandidat begann, konnte dieser sich mit der Steuerung und Eigenheiten des jeweiligen Videoplayers anhand eines Testvideos vertraut machen. In der folgenden Tabelle sind alle verwendeten Videos aufgelistet, mit zusätzlichen Angaben zur Dauer, sowie bei welchen Aufgaben das Video zum Einsatz kam.

Video	Thema	Dauer	Aufgabentyp
1_documentary	Naturdokumentation	00:50:04	Zielsuche
2_documentary		00:44:32	
1_news	Nachrichtensendung	00:53:14	Zielsuche
2_news		00:53:30	
1_football	Ausschnitt eines Fußballspiels	00:15:02	Zählaufgaben
2_football		00:15:00	
1_gameshow	Ausschnitt einer Quizshow	00:13:38	Zählaufgaben
2_gameshow		00:15:01	

Tabelle 6.1: Verwendete Videos für die Evaluierung

Die erste Aufgabe für den Benutzer war die Zielsuchaufgabe mit Version 1 der Dokumentation, danach dieselbe Aufgabe mit Version 1 des Nachrichten Videos. Die Aufgabe war beendet, sobald der Benutzer glaubte die richtige Stelle im Video gefunden zu haben, die ihm zuvor in der kurzen 20 sekundenlangen Sequenz gezeigt wurde. Direkt im Anschluss wurden die Aufgaben gestellt, bei denen man bestimmte Ereignisse zählen musste. Bei dem ersten Video, welches ca. 15 Minuten lang war, musste der Benutzer in dem Fußball Video 1 vier verschiedene Aktionen zählen, die beliebig oft vorkommen konnten. Diese waren im Einzelnen Tore, gelbe beziehungsweise rote Karten, Einwürfe und Eckstöße. Von welcher Mannschaft die Aktionen ausgingen, war nicht relevant. In dem zweiten Testvideo wurde eine Quizshow gezeigt, bei der zu einer Frage vier Antwortmöglichkeiten gegeben wurden, wobei immer nur eine richtig war und diese, nach

beantworten durch die Showkandidaten, grün aufgeleuchtet sind. Der Studienteilnehmer musste bei diesem Video immer die richtigen Antworten zählen, die innerhalb der 15 Minuten gezeigt wurden. Nachdem alle Aufgaben der ersten Applikation vom Nutzer erledigt waren, wurde der Proband gebeten einen Fragebogen auszufüllen, welcher nach dem NASA-TLX (*Task Load Index*) [77] als Vorbild gestaltet wurde. Dieser wurde noch um Fragen zur Erfassung von demografischen Daten und Bewertung des zuvor verwendeten Videoplayers erweitert. Im Anschluss darauf wiederholte der erste Benutzer diese Aufgaben, musste hierbei jedoch den Standardvideoplayer benutzen und bekam immer die Version der Videos vorgelegt, welche noch nicht Teil der Aufgaben war. Diese unterschieden sich nur vom Inhalt zur vorherigen Version, jedoch nicht vom Typ. So gab es beispielsweise zwei Ausschnitte der Quizshow, jeweils mit unterschiedlichen Kandidaten und Fragen. Auch hier wurde nach Beenden der Aufgaben der gleiche Fragebogen ausgeteilt, der um die Frage ergänzt wurde, welcher Videoplayer dem Benutzer besser gefallen hat.

6.3 Evaluierung

Die Evaluierung der Daten wird verschiedene Hypothesen überprüfen. Zum einen wird versucht mit statistischen Mitteln herauszufinden, welcher Videoplayer sich besser geeignet, um die geforderten Aufgaben zu lösen. Ein Messwert dafür ist die benötigte Zeit und mit welcher Genauigkeit die Probanden die Suchaufgaben fertiggestellt haben. Da der rein quantitative Aspekt allein keine allgemeingültige Aussage auf die Gesamtleistung der Videoplayer geben kann, wird als Teil der Studie auch die subjektive Wahrnehmung mit einbezogen. Dies geschieht auf Basis der Daten, die mit Hilfe des Fragebogens gesammelt wurden.

6.3.1 Zielsuche

Bei den Zielsuchaufgaben ging es darum die gesuchte Szene so schnell wie möglich zu finden. Dabei war es auch wichtig, ob die Szene vom Probanden letztendlich gefunden wurde. Dieser Messwert kann jedoch vernachlässigt werden, da jeder Nutzer bei jeder Zielsuchaufgabe die entsprechenden Szenen mit beiden Videoplayers gefunden hat. Aus diesem Grund kann der Fokus bei der Evaluierung rein auf die Suchdauer gelegt werden. Die Studienteilnehmer konnten mit Hilfe des Flickplayers die gesuchte Stelle im Nachrichten Video mit einer Durchschnittszeit von 142,31 Sekunden finden, bei einer Standardabweichung von $\pm 58,72$. Im Vergleich dazu konnten die Nutzer mit dem Standardplayer diese Aufgaben im Schnitt innerhalb 81,31 Sekunden beenden und das mit einer Standardabweichung von $\pm 30,34$. In Abbildung 6.3 ist der Boxplot zu den Such-

zeiten grafisch dargestellt. Hier ist leicht zu erkennen, dass die Suchzeiten des Standardplayers, verglichen zu denen des Flickplayers, deutlich geringer sind. Die Boxplotlänge ist ebenfalls viel geringer beim Standardplayer, was bedeutet, dass die Suchzeiten generell nicht besonders stark von einander abweichen. Ob die Daten statistisch gesehen einen signifikanten Unterschied aufweisen, wird mit Hilfe des Wilcoxon-Vorzeichen-Rang-Test untersucht [78]. Dieser lässt sich anhand von Wertepaaren durchführen, was sich für den direkten Vergleich der Videoplayer hervorragend dazu eignet. Auswertung bei den Nachrichten hat ergeben, dass bei $z = -3,1284$, $p = 0,05$ und die Summe der positiven Ränge von 128,5, sowie die Summe der negativen Ränge von 7,5, der Standardplayer im Vergleich zum Flickplayer signifikant schneller war. Das nächste Video, welches als Teil der

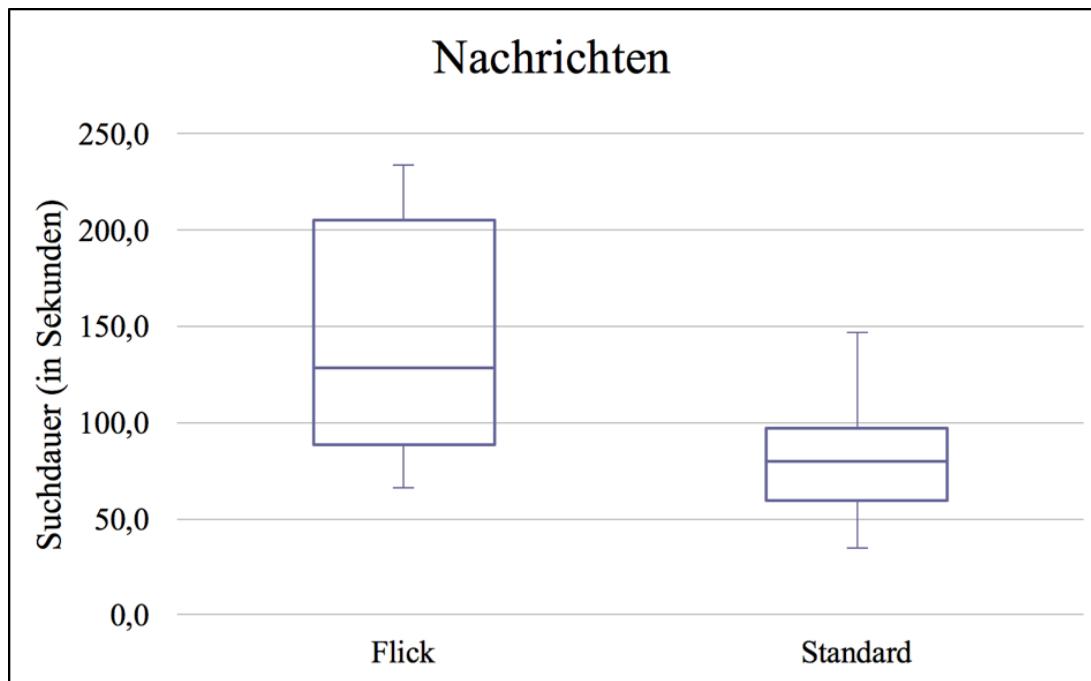


Abbildung 6.3: Boxplot zur Suchdauer Nachrichten Videos

Untersuchung herangezogen wurde, war die Dokumentation. Auf Basis des Mittelwertes lässt sich hier ein großer Unterschied erkennen, denn die durchschnittlichen Suchzeiten beim Flickplayer liegen hier bei 128,81 Sekunden, bei einer Standardabweichung von $\pm 90,33$. Die Durchschnittszeiten beim Standardplayer liegen mit 67,81 Sekunden mit einer Standardabweichung von $\pm 29,3$. Wenn man jedoch den Boxplot in Abbildung 6.4 genauer betrachtet, dann fällt auf, dass die Boxplotlänge, sowie dessen Position für beide Videoplayer eine ähnliche Lage haben, also der Großteil der Teilnehmer die Aufgaben in einem ähnlichen Zeitfenster abgeschlossen haben. Der Boxplot des Flickplayers zeigt

aber, dass es Ausreißer gab, die sich negativ auf die durchschnittliche Performanz der Applikation ausgewirkt haben. Auch hier wird wieder die Signifikanz mit dem Wilcoxon-Vorzeichen-Rang-Test untersucht. Der gelieferten Ergebnisse sind $z = -2,7923$ bei $p = 0,05$ und einer Summe der positiven Ränge von 122, bzw. der negativen Ränge von 14. Diese Ergebnisse lassen darauf schließen, dass es einen Unterschied gibt und dieser auch signifikant ist. Das bedeutet, dass auch hier der Standardplayer schneller als der Flickplayer war. Daraus lässt sich schließen, dass bei langen Videos die Suche von einer bestimmten Szene mit dem Standardplayer, im Vergleich zur Suche mit dem Flickplayer, schneller ist. Da dies die reinen statistischen Werte über die Performanz des Flickplayer sind, lassen sich damit keine Aussagen darüber treffen, wie die Testpersonen die Navigationsmethode subjektiv empfunden haben. Dies wird mit der Analyse der Fragebögen im weiteren Verlauf der Arbeit erörtert.

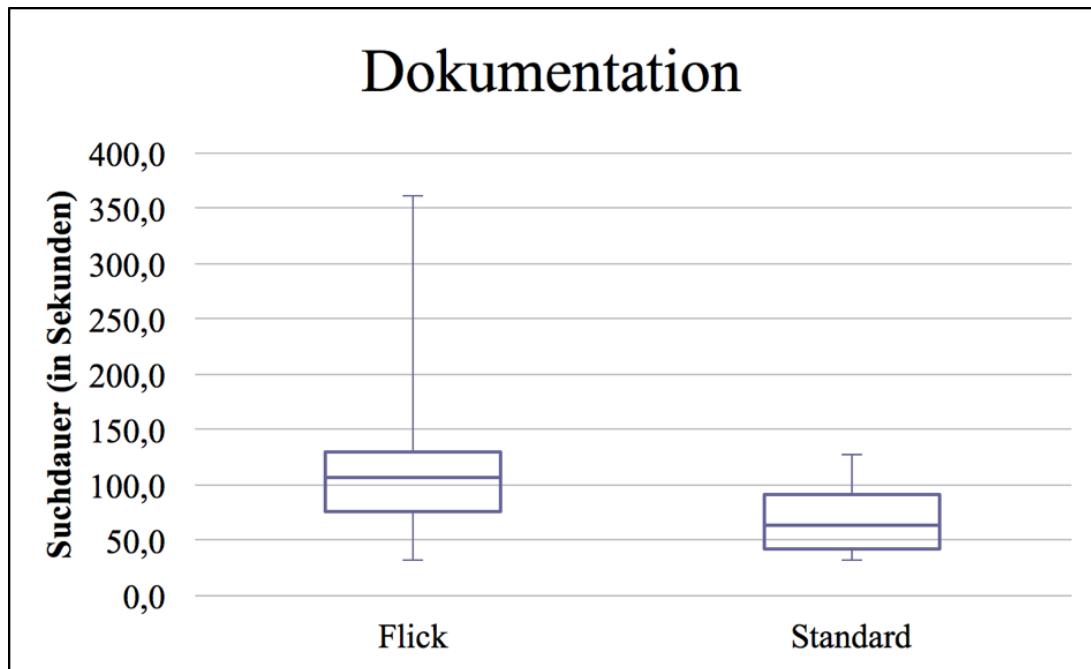


Abbildung 6.4: Boxplot zur Suchdauer Dokumentation Videos

Die gesammelten Daten von den Zählaufgaben werden im nächsten Kapitel analysiert und evaluiert.

6.3.2 Zählaufgaben

Bei den Zählaufgaben mussten die Nutzer ohne zeitliche Beschränkung die vorgegeben Ereignisse Zählen. Relevant waren hier für die Evaluierung zwei Faktoren, welche zum einen die benötigte Zeit war und zum anderen die Genauigkeit der Zählungen. Das erste Video war der Ausschnitt aus der Wer wird Millionär Sendung. Die gesuchten Ereignisse waren die korrekten Antworten, also A, B, C oder D. Die durchschnittliche Zeit, die benötigt wurde, war beim Flickplayer 191,56 Sekunden mit einer Standardabweichung von $\pm 106,83$. Im Vergleich liegt die Zeit mit dem Standardplayer im Schnitt bei 140,81 Sekunden mit $\pm 108,07$. Wenn man den Boxplot aus Abbildung 6.5 betrachtet, dann fällt auf, dass die meisten Nutzer die Aufgaben mit dem Standardplayer in einer ähnlichen Zeit beendet haben, jedoch Ausreißer mit ca. 450 Sekunden Suchdauer den Durchschnitt stark beeinflusst haben. Die Signifikanz wurde wieder mit dem Wilcoxon-Vorzeichen-Rang-Test ermittelt. Die Werte $z = -2,6371$ bei $p = 0,05$ und die Summe der positiven Ränge mit 119, bzw. die Summe der negativen Ränge mit 17 lassen daran schließen, dass es einen signifikanten Unterschied zwischen den Videoplayern gibt. Dieser Unterschied geht auch hier wieder zugunsten des Standardplayers. Das nächste

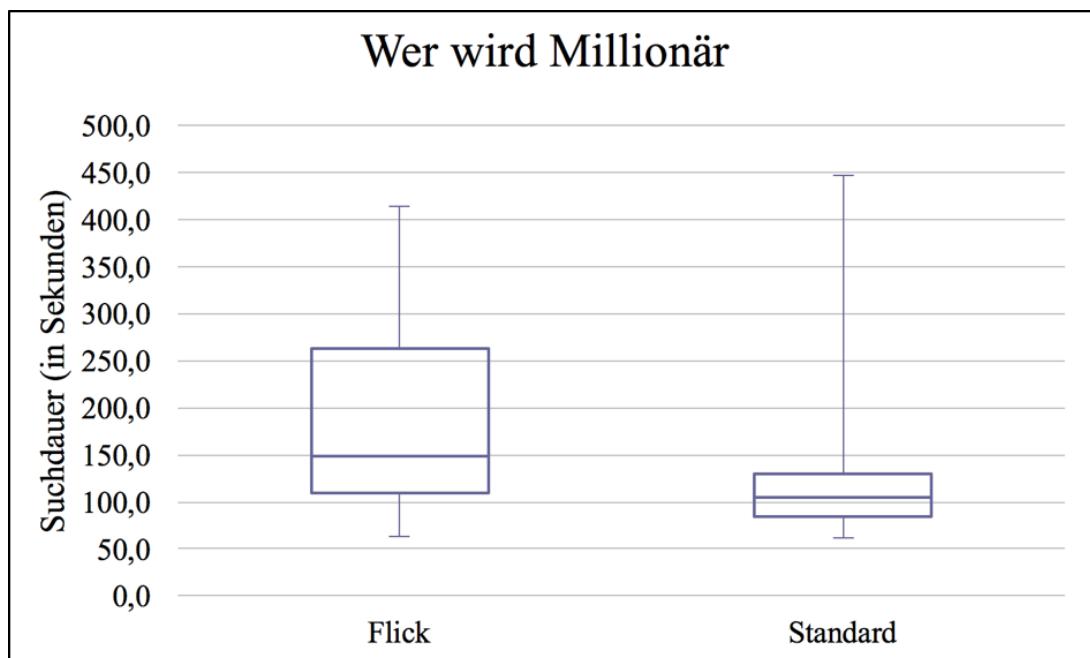


Abbildung 6.5: Boxplot zur Suchdauer Quizshow Videos

Video ist der Ausschnitt des Fußballspiels. Wie schon in einer der vorherigen Kapiteln erwähnt, mussten hier Ereignisse wie Mahnkarten, Tore, Einwürfe und Eckstöße gezählt

werden. Die Durchschnittszeit beim Flickplayer liegt bei 336 Sekunden mit einer Standardabweichung von $\pm 166,96$. Die Probanden haben die Aufgaben mit dem Standardplayer in einer mittleren Zeit von 305,81 Sekunden bewältigt. Die Standardabweichung liegt bei dieser Messung bei $\pm 202,85$. Die Boxplots in Abbildung 6.6 weisen Ähnlichkeiten bezüglich Form und Position auf. Wie die Durchschnittswerte schon angedeutet haben, sind die Suchzeiten bei dem Standardplayer leicht verkürzt. Die Resultate aus dem Wilcoxon-Vorzeichen-Rang-Test besagen mit einem p-Wert von 0,28434, welcher das Signifikanz-Niveau von $p = 0,05$ übersteigt, dass die Unterschiede nicht signifikant sind. Wie die Auswertung gezeigt hat, kann der Flickplayer den Nutzer nicht dabei unterstützen die Zählaufgaben schneller als mit dem Standardplayer zu erledigen, sondern unterliegt diesem. Eine weitere Annahme der Studie war, dass der Nutzer bei der Navigation mit dem Flickplayer mehr von dem Inhalt eines Video mitbekommt, als mit herkömmlicher Navigation. Daher sollte auch die Richtigkeit bzw. Genauigkeit der Zählwerte der Nutzer überprüft werden. Dafür wurde ein gewichtetes Modell erstellt, welches so funktioniert, dass die Anzahl der Nutzer ermittelt wird, welche alle Ereignisse gefunden haben oder einen Fehler, zwei Fehler oder mehr als zwei Fehler gemacht haben. Für jeden Nutzer, der die Aufgaben fehlerfrei beendet hat, bekommt der jeweilige Videoplayer vier Punkte, für einen Fehler wird ein Punkt abgezogen, bei zwei Fehlern wird ein weiterer Punkt abgezogen, sodass bei mehr als zwei Fehlern nur noch ein Punkt vergeben wird. Somit ergibt sich eine maximale Punktzahl von 64 Punkten bei 16 Probanden. Die erreichten Punkten werden im Anschluss durch die maximale Punktzahl geteilt, sodass sich ein Prozentwert erhalten wird, welcher die Genauigkeit des jeweiligen Videoplayers repräsentiert. Der Flickplayer hat bei den Quizshow insgesamt 14 Nutzer gehabt, die die Aufgaben fehlerfrei beendet haben. Eine Gesamtpunktzahl von 61 wurde erreicht, was einer Genauigkeit von 95,31% entspricht. Im Vergleich dazu hat der Standardplayer nur 52 Punkte erreicht, da nur 12 Personen alle Ereignisse richtig gezählt haben. Die Genauigkeit ist daher verglichen zum Flickplayer mit 84,36% geringer. Der Schwierigkeitsgrad von den Fußball-Videos ist im Allgemeinen höher, da es weniger ersichtlich ist, wann und wo ein Ereignis geschieht. Des Weiteren ist aufgrund von Wiederholungen bei einer schnellen Navigation nicht immer ganz klar, ob das selbe Ereignis mehrfach gezeigt wird, oder es sich um ein neues handelt. Die Gesamtpunktzahl, die durch den Flickplayer erreicht wurde, liegt bei 49, da insgesamt nur 7 Personen alle Ereignisse richtig gezählt haben. Somit wurden insgesamt 76,56% der maximalen Punkte erreicht. Dem gegenübergestellt steht der Standardplayer, welcher insgesamt nur 28 Punkte erreicht hat. Nur eine Personen hat fehlerfrei gezählt und 13 Personen hatten zwei oder mehr Fehler. Somit liegt die Genauigkeit bei 43,75% was deutlich ungenauer als beim Flickplaner ist. Die Studie hat somit bestätigt, dass die Teilnehmer mit dem Flickplayer zwar länger gebraucht haben, um die Zählaufgaben zu lösen, jedoch mehr vom Inhalt

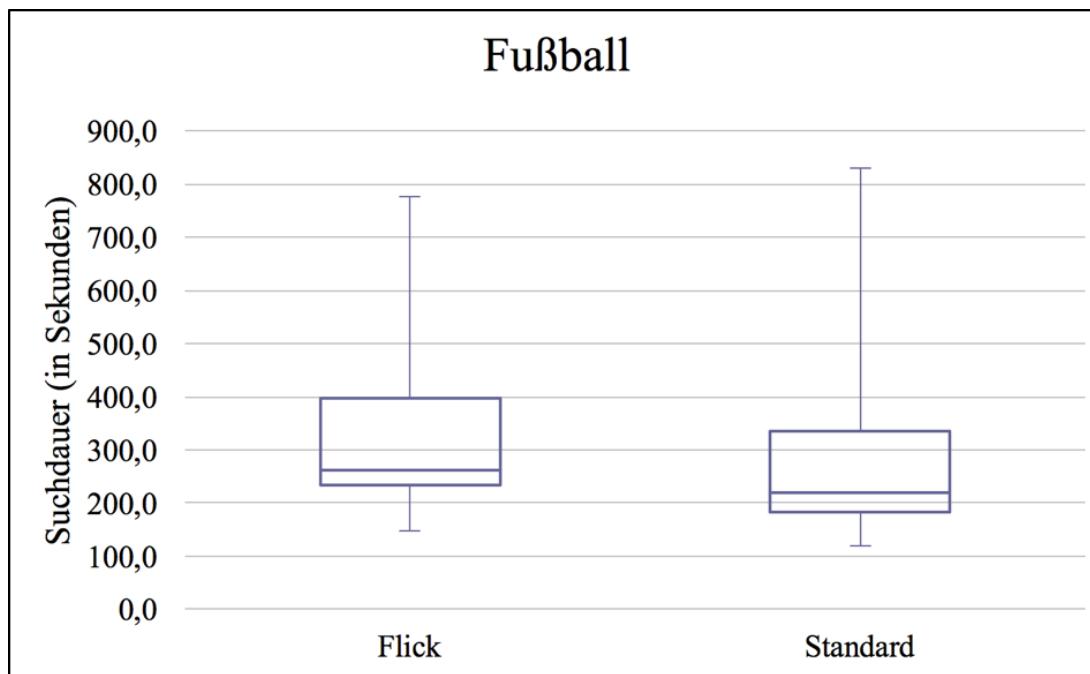


Abbildung 6.6: Boxplot zur Suchdauer Fußball Videos

aufgenommen haben.

Im folgenden Kapitel wird die subjektive Wahrnehmung der Probanden untersucht, um festzustellen, ob die Navigationsmethode des Flickplayers, im Vergleich zum Standardplayer mit Zeitleistennavigation, einen positiveren Eindruck gegeben hat.

6.3.3 Benutzerbewertung im NASA TLX

Wie Kapitel schon 6.2 schon erwähnt wurde, basiert der erste Teil des Fragebogens auf dem NASA-TLX, welcher häufig eingesetzt wird, um die Effektivität von Systemen zu testen. Dabei werden insgesamt sechs Fragen gestellt, von der jede eine Bewertung auf einer Intervallskala von 1 bis 21 vorgibt, wobei 1 immer eine niedrigere Belastung und 21 das andere Extrem bedeutet. Im Falle von Frage 4 bedeutet 1, dass die Aufgabe perfekt gelöst wurde und 21, dass der Nutzer gescheitert ist. Die Fragen lauten im Einzelnen wie folgt:

1. Wie anspruchsvoll waren die Aufgaben auf mentaler Ebene?
2. Wie körperlich anspruchsvoll waren die Aufgaben?
3. Wie hoch war der zeitliche Druck beim Erledigen der Aufgaben?

4. Wie erfolgreich haben Sie die gestellten Aufgaben erledigt?
5. Wie stark mussten Sie sich anstrengen, um die Aufgaben zu erledigen?
6. Wie unsicher, irritiert, gestresst, entmutigt und genervt waren Sie bei der Durchführung der Aufgaben?

Alle Probanden haben diesen Fragebogen für jeden Videoplayer einmal ausgefüllt und zwar immer direkt nach dem Erledigen der Aufgaben mit der jeweiligen Applikation. Der Grund hierfür war, dass die Eindrücke so am frischsten sind. Um nun herauszufinden, ob einer der Videoplayer statistisch gesehen besser ist, als der andere, wurde für jede einzelne Frage der Wilcoxon-Vorzeichen-Rang-Test durchgeführt. Dies bietet sich an, da sich die gelieferten Werte zu Stichprobenpaare kombinieren lassen, um auf deren Basis festzustellen, ob es einen signifikanten Unterschied zwischen den beiden Videoplayern gibt. In Abbildung 6.7 sind die durchschnittlichen Bewertungen durch die Teilnehmer zu

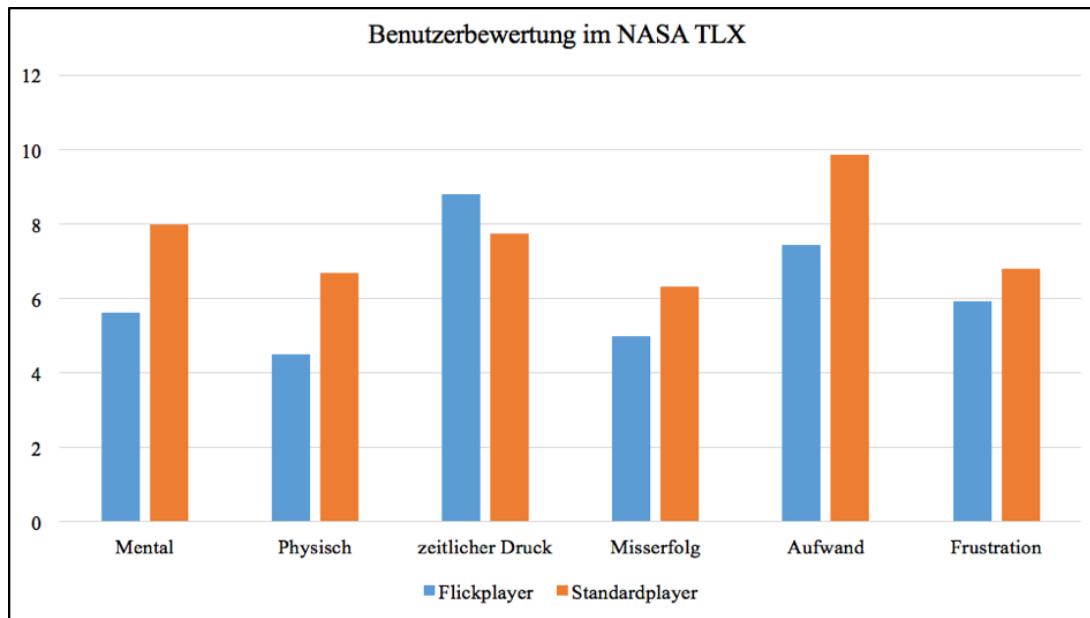


Abbildung 6.7: NASA TLX Bewertungsverteilung von 1 - 21

sehen. Ein niedriger Wert bedeutet, dass die Belastung bspw. eher geringer war, was ein positiver Faktor für einen Videoplayer ist. Man kann auf dem Diagramm schnell erkennen, dass der Flickplayer im Vergleich zum Standardplayer bei den meisten Dimensionen eine im Durchschnitt besser Bewertung erhalten hat. Der zeitliche Druck beim Ausführen der Aufgaben war der einzige Faktor, bei dem der Standardplayer scheinbar einen

Vorteil hatte. Die Auswertung des Wilcoxon-Vorzeichen-Rang-Test ergab, dass die Unterschiede beider Stichproben bei den Fragen bezüglich des zeitlichen Drucks und der Frustration mit $z = -0,3844$ und $z = -1,0225$ zwar gegeben ist, dieser aber bei $p = 0,05$ nicht signifikant war. Die weitere Untersuchung auf die Frage der mentalen Belastung hat ergeben, dass bei $z = -2,1344$ und $p = 0,05$ der Flickplayer eine geringere Belastung hat und der Unterschied signifikant ist. Die physische Belastung hat ebenfalls einen signifikanten Unterschied zugunsten des Flickplayers mit $z = -2,4006$ und $p = 0,05$. Bei den Fragen 4 und 5 zum Thema Misserfolg und betriebener Aufwand hat auch hier wieder der Flickplayer einen signifikanten Unterschied im Vergleich zum Standardplayer, mit $z = -2,1972$ und $z = -2,6052$ mit jeweils und $p = 0,05$.

6.3.4 Bevorzugter Videoplayer

Ein Teil des Fragebogens war es auch herauszufinden, welcher Videoplayer den Nutzern subjektiv besser gefallen hat. Um dies zu ermitteln, sollten die Teilnehmer zum einen beide Videoplayer anhand einer Schulnotenskala von 1 – 6 bewerten, wobei 1 die beste Note und 6 die schlechteste Note ist. Des Weiteren gab es ein Textfeld, in dem die Nutzer weitere Kommentare schreiben konnten, um explizitere Aussagen treffen zu können. Die Abbildung 6.8 zeigt die Verteilung der einzelnen Benotungen durch die Probanden. Die blaue Linie repräsentiert die Noten des Flickplayers und die orangene Linie die des Standardplayers. Hier ist leicht zu erkennen, dass die Noten des Flickplayers in den meisten Fällen besser ist, im Vergleich zu den Bewertungen des Standardplayers. Im Durchschnitt haben die 16 Teilnehmer eine Note von 1,75 für den Flickplayer geben, mit einer Standardabweichung von $\pm 0,66$ und für den Standardplayer eine Durchschnittsnote von 2,5 mit einer Standardabweichung von $\pm 0,7$. In 11 von 16 Fällen haben die Probanden den Flickplayer dem Standardplayer eindeutig vorgezogen, was 68,75% entspricht. Allerdings muss hier hervorgehoben werden, dass in den anderen fünf Fällen der Standardplayer nicht immer als präferierter Videoplayer genannt werden kann, sondern in drei Fällen beide Videoplayer eine gleiche Benotung erhalten haben und nur zwei Probanden den Standardplayer vorgezogen haben. Die Auswertung der schriftlichen Anmerkungen hat ergeben, dass die Nutzer insbesondere die permanente Interaktion mit der Navigationsleiste beim Standardplayer negativ bewerten. Sie mussten sich sehr darauf konzentrieren, dass sie den Schieberegler nicht zu schnell bewegen und somit die Aufmerksamkeit nicht auf den Inhalt des Videos gerichtet werden konnte. Des Weiteren wurde bemängelt, dass die Position der Navigationsleiste am oberen Bildschirmrand ist und dadurch die Hand während der Interaktion das Video bedeckt und eine schiefe Körperhaltung erforderlich wurde. Im Gegensatz dazu wurde dies beim Flickplaner positiv hervorgehoben, dass die Folgen einer Wischgeste auch dann noch einen Einfluss auf die Navigation haben, wenn

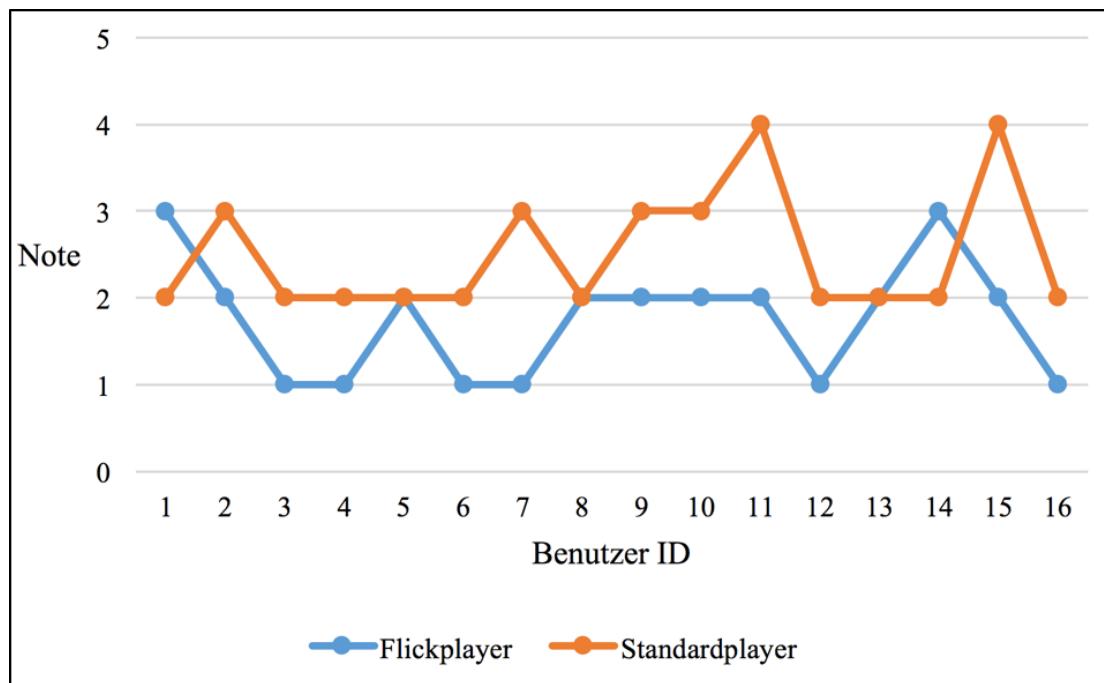


Abbildung 6.8: Subjektive Bewertung der Videoplayer

die Hand vom Bildschirm genommen wird. Dadurch war der Gesamteindruck, dass die Navigation auf dem Flickplaner, in Bezug auf den körperlichen Aufwand weniger, anstrengend ist.

Das folgende Kapitel bildet mit der Schlussfolgerung und dem Ausblick für die Zukunft den Schluss dieser Arbeit.

In der heutigen Zeit zeigt der Konsum von Videos eine enorm steigende Entwicklung. Smartphones und Tablets tragen hierbei eine erhebliche Rolle für das Wachsen von privaten Video-Bibliotheken. Die Herausforderung ist es diese Videos und Sammlungen für die Nutzer zugänglich zu machen, sodass diese auch wiedergefunden werden. Die vorliegende Arbeit untersuchte die Thematik von Videobrowsing insbesondere im mobilen Sektor. Einige Applikationen für den Desktop-Bereich sind bereits erfolgreich für Tablets und Smartphones angepasst worden, jedoch gibt es keine weitverbreitete Alternativen zu Videoplayern mit einer zeitbasierten Suchleiste. Leichte Anpassungen wie Vorschaubilder auf der Navigationsleiste sind jedoch keine Seltenheit bei bekannten Videoportalen wie YouTube.

Auf der Grundlage von existierenden mobilen Videoplayern, wie dem SwiPlayer, wurde eine Videoapplikation vorgeschlagen, welche mit Hilfe von Wischgesten ein Video in stark beschleunigter Abspielrate wiedergibt. Dadurch sollte es für den Betrachter einfacher sein, schnell einen Überblick über den Inhalt des Videos zu erhalten. Aufgrund der Granularität von mehreren Geschwindigkeitsstufen, die eng mit der Wischgesteninteraktion verbunden sind, sollte dem Nutzer ein positives Benutzererlebnis geboten werden.

Bei der Entwicklung des Flickplayers lagen die Hauptprobleme vor allem bei den eingeschränkten Ressourcen. Ein Video in bis zu 32-facher Geschwindigkeit abzuspielen, ohne auf eine flüssige Wiedergabe zu verzichten, war nicht möglich. Daher mussten Wege gefunden werden, dieses Problem zu kaschieren. Eine für den Produktivgebrauch nicht geeignete Lösung war es, ein Video in drei verschiedenen Geschwindigkeiten zu kodieren und diese übereinanderzulegen. Somit konnte durch schnelles Schalten zwischen den Videos und kontinuierliches Synchronisieren den Eindruck vermittelt werden, dass es sich nur um ein einzelnes Video handelt. Da das Ziel dieser Arbeit die Entwicklung und Evaluierung der Wischgesteninteraktion ist, konnte dieser Kompromiss eingegangen werden.

Die Fragestellung, ob eine derartige Navigationsmethode von Vorteil ist, wurde im

Rahmen einer Benutzerstudie mit 16 Teilnehmern ermittelt. Verschiedene Aufgaben aus den Bereichen Zielsuche und Zählsuche sollten Aufschluss darauf geben, ob der Flickplayer in Gegenüberstellung zu einem Standardplayer eine bessere Unterstützung bietet. Die Analyse der gewonnenen Daten hat ergeben, dass in fast allen Fällen die Probanden mit dem Standardplayer die Aufgaben signifikant schneller fertiggestellt haben. Dies war sowohl bei den Zielsuchaufgaben, sowie bei der Zählsuche der Fall. Zusätzlich wurde bei den Zählsuchaufgaben auch die Genauigkeit der gezählten Ereignisse ausgewertet. Hier wurde sehr deutlich, dass die Nutzer mit dem Flickplayer genauere Endergebnisse erzielen konnten, insgesamt die Ereignisse besser erkannt und sich weniger häufig verzählt haben. Als Teil der Benutzerstudie hat jeder Teilnehmer im Anschluss für beide Videoplayer Applikationen einen Bewertungsbogen ausgefüllt, welcher auf dem NASA TLX Fragebogen basiert. Die gewonnenen Daten besagen, dass die Nutzer mit dem Flickplayer eine geringere mentale und physische Belastung erlebten. Ebenso das Frustniveau und das Empfinden von Misserfolg konnte der Flickplayer, verglichen mit dem Standardvideoplayer, auf einem niedrigeren Niveau halten. Eine subjektive Benotung durch die Teilnehmer ergab, dass fast 68% den Flickplayer für das Bearbeiten der Aufgaben bevorzugten, obwohl sie damit mehr Zeit benötigten. Besonders hervorgehoben wurde, dass die Wischgesteninteraktion intuitiv sei und es dem Nutzer einfacher mache, sich auf den Inhalt der Videos zu konzentrieren.

Auf Grundlage der gewonnenen Erkenntnisse durch die Studie kann gesagt werden, dass diese spezielle Wischgesteninteraktion als Navigationsmethode vom Nutzer akzeptiert wird, jedoch keine kürzeren Suchzeiten bewirkte. In Zukunft sollte daher weitere Forschung in diesem Bereich betrieben werden, bei der die Konzepte von traditionellen Navigationsleisten mit Wischgesteninteraktion kombiniert werden. Somit könnte das Benutzererlebnis für die breite Masse verbessert werden, ohne Einbüßen bei der Suchgeschwindigkeit zu machen.

Literaturverzeichnis

- [1] Statista GmbH. Die Opfer des smartphone-booms. <http://de.statista.com/infografik/1958/geraete-absatz-im-bereich-consumer-electronics/>. Eingesehen am 02.08.2015.
- [2] Statista GmbH. Absatz von tablets weltweit vom 2. quartal 2010 bis zum 3. quartal 2015. <http://de.statista.com/statistik/daten/studie/181569/umfrage/weltweiter-absatz-von-media-tablets-nach-quartalen/>. Eingesehen am 27.08.2015.
- [3] International Data Corporation. Idc worldwide tablet forecast. <http://www.idc.com/getdoc.jsp?containerId=prUS25480015>. Eingesehen am 17.02.2016.
- [4] Adriana Reveiu, Marian Dardala, and Felix Furtuna. Using content-based multimedia data retrieval for multimedia content adaptation. In *Human-Computer Interaction. HCI Intelligent Multimodal Interaction Environments*, pages 486–492. Springer, 2007.
- [5] Yovisto. Yovisto homepage. <http://www.yovisto.com>. Eingesehen am 28.10.2015.
- [6] Nick Harris. *Beginning IOS Programming: Building and Deploying IOS Applications*. John Wiley & Sons, 2014. ISBN 9781118841440.
- [7] Apple Inc. Apple ios developer library: Key-value observing. <https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/KeyValueObserving/KeyValueObserving.html>. Eingesehen am 03.10.2015.
- [8] Apple Inc. Apple ios developer library: About avfoundation. https://developer.apple.com/library/prerelease/ios/documentation/AudioVideo/Conceptual/AVFoundationPG/Articles/00_Introduction.html. Eingesehen am 03.10.2015.
- [9] Apple Inc. Avfoundation playback. https://developer.apple.com/library/prerelease/ios/documentation/AudioVideo/Conceptual/AVFoundationPG/Articles/02_Playback.html. Eingesehen am 03.10.2015.

- [10] Apple Inc. Apple ios developer library: Uiresponder chain. https://developer.apple.com/library/ios/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/event_delivery_responder_chain/event_delivery_responder_chain.html. Eingesehen am 05.10.2015.
- [11] Klaus Schoeffmann, Mario Taschwer, and Laszlo Boeszoermenyi. The video explorer: a tool for navigation and searching within a single video based on fast content analysis. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 247–258. ACM, 2010.
- [12] Gerald Friedland, Luke Gottlieb, and Adam Janin. Joke-o-mat: browsing sitcoms punchline by punchline. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 1115–1116. ACM, 2009.
- [13] Manfred Del Fabro, Klaus Schoeffmann, and Laszlo Böszörmenyi. Instant video browsing: a tool for fast non-sequential hierarchical video browsing. In *Proceedings of the 6th international conference on HCI in work and learning, life and leisure: workgroup human-computer interaction and usability engineering*, pages 443–446. Springer-Verlag, 2010.
- [14] Leif Azzopardi, Douglas Dowie, and Kelly Ann Marshall. Yoosee: A video browsing application for young children. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 1017–1017. ACM, 2012.
- [15] Apple Inc. ios human interface guidelines: Interactivity and feedback. <https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/InteractivityInput.html>. Eingesehen am 20.02.2016.
- [16] Marco A Hudelist, Klaus Schoeffmann, and Laszlo Boeszoermenyi. Mobile video browsing with the thumbbrowser. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 405–406. ACM, 2013.
- [17] Thorsten Karrer, Moritz Wittenhagen, and Jan Borchers. Pocketdragon: a direct manipulation video navigation interface for mobile devices. In *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, page 47. ACM, 2009.

- [18] Marco A Hudelist, Klaus Schoeffmann, and Qing Xu. Improving interactive known-item search in video with the keyframe navigation tree. In *MultiMedia Modeling*, pages 306–317. Springer, 2015.
- [19] Wolfgang Huerst, Georg Goetz, and Martina Welte. Interactive video browsing on mobile devices. In *Proceedings of the 15th international conference on Multimedia*, pages 247–256. ACM, 2007.
- [20] Jochen Huber, Juergenrgen Steimle, Roman Lissermann, Simon Olberding, and Max Muehlhaeuser. Wipe’n’watch: spatial interaction techniques for interrelated video collections on mobile devices. In *Proceedings of the 24th BCS Interaction Specialist Group Conference*, pages 423–427. British Computer Society, 2010.
- [21] Klaus Schoeffmann, Kevin Chromik, and Laszlo Boeszoermenyi. Video navigation on tablets with multi-touch gestures. In *Multimedia and Expo Workshops (IC-MEW), 2014 IEEE International Conference on*, pages 1–6. IEEE, 2014.
- [22] Wolfgang Huerst and Konrad Meier. Interfaces for timeline-based mobile video browsing. In *Proceedings of the 16th ACM international conference on Multimedia*, pages 469–478. ACM, 2008.
- [23] Huahui Wu, Mark Claypool, and Robert Kinicki. Guidelines for selecting practical mpeg group of pictures. 2005.
- [24] Klaus Schoeffmann. The stack-of-rings interface for large-scale image browsing on mobile touch devices. In *Proceedings of the ACM International Conference on Multimedia*, pages 1097–1100. ACM, 2014.
- [25] Borivoje Furht. *Handbook of multimedia for digital entertainment and arts*. Springer, 2010. ISBN 9780387890241.
- [26] Yajin Zhou, Xinwen Zhang, Xuxian Jiang, and Vincent W Freeh. Taming information-stealing smartphone applications (on android). In *Trust and Trustworthy Computing*, pages 93–107. Springer, 2011.
- [27] International Data Corporation. Idc worldwide quarterly tablet tracker. <http://www.idc.com/getdoc.jsp?containerId=prUS25989015>. Eingesehen am 29.10.2015.
- [28] Nikolaus Fischer and Stefan Smolnik. Das potential von tablets in der universitaeren lehre. In *DeLFi*, pages 237–242, 2012.

- [29] Ralf Steinmetz. *Multimedia-Technologie: Grundlagen, Komponenten und Systeme*. Springer-Verlag, 2013. ISBN 9783642583230.
- [30] Kari Kallinen, Jan Kallenbach, and Niklas Ravaja. The effects of content type and presentation style on user experiences of multimedia content on a tablet pc. In *Human-Computer Interaction. Users and Applications*, pages 466–475. Springer, 2011.
- [31] Kenton O’Hara, April Slayden Mitchell, and Alex Vorbau. Consuming video on mobile devices. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 857–866. ACM, 2007.
- [32] Lingfen Sun and Ali Alfayly. Qoe-driven management schemes for multimedia services. *E-LETTER*, 2015.
- [33] Klaus Schoeffmann, David Ahlstrom, Marco Hudelist, et al. 3-d interfaces to improve the performance of visual known-item search. *Multimedia, IEEE Transactions on*, 16(7):1942–1951, 2014.
- [34] Birgit Gaiser. *Good tags-bad tags*, volume 47. Waxmann Verlag, 2008. ISBN 9783830920397.
- [35] Thomas Sillmann. *Apps fuer iOS 9 professionell entwickeln: Sauberen Code schreiben mit Objective-C und Swift. Stabile Apps programmieren. Techniken und Methoden von Grund auf verstehen*. Hanser, 2015.
- [36] Apple Inc. Apple ios developer library: Working with protocols. <https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/WorkingwithProtocols/WorkingwithProtocols.html>. Eingesehen am 02.10.2015.
- [37] Apple Inc. Apple ios developer library: Nsnotificationcenter class. https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSNotificationCenter_Class/. Eingesehen am 02.10.2015.
- [38] Carlo Chung and James Bucanek. *Pro Objective-C Design Patterns for iOS*. Springer, 2011. ISBN 9781430233305.
- [39] James Bucanek. *Learn iOS 8 App Development*. Apress, 2014. ISBN 9781484202081.
- [40] Bob McCune. *Learning AV Foundation: A Hands-on Guide to Mastering the AV Foundation Framework*. Pearson Education, 2014. ISBN 9780133563825.

- [41] Apple Inc. Apple ios developer library: Uiresponder class. https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIResponder_Class/. Eingesehen am 05.10.2015.
- [42] Apple Inc. Apple ios developer library: UIGesturerecognizer class. https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIGestureRecognizer_Class/. Eingesehen am 05.10.2015.
- [43] Kai Juengling, Scott Blunsden, and Cristina Versino. Videozoom: An interactive system for video summarization, browsing and retrieval. In *Advances in Visual Computing*, pages 323–332. Springer, 2014.
- [44] Xin Fan, Xing Xie, Wei-Ying Ma, Hong-Jiang Zhang, and He-Qin Zhou. Visual attention based image browsing on mobile devices. In *Multimedia and Expo, 2003. ICME'03. Proceedings. 2003 International Conference on*, volume 1, pages I–53. IEEE, 2003.
- [45] Yuan He and Yunhao Liu. Supporting vcr in peer-to-peer video-on-demand. In *Network Protocols, 2007. ICNP 2007. IEEE International Conference on*, pages 328–329. IEEE, 2007.
- [46] Joydeep Ghosh, Yong Jae Lee, and Kristen Grauman. Discovering important people and objects for egocentric video summarization. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1346–1353. IEEE, 2012.
- [47] Chong-Wah Ngo, Yu-Fei Ma, and Hong-Jiang Zhang. Video summarization and scene detection by graph modeling. *Circuits and Systems for Video Technology, IEEE Transactions on*, 15(2):296–305, 2005.
- [48] Nuno Vasconcelos and Andrew Lippman. Bayesian modeling of video editing and structure: Semantic features for video summarization and browsing. In *Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on*, pages 153–157. IEEE, 1998.
- [49] Hongliang Bai, Lezi Wang, Yuan Dong, and Kun Tao. Interactive video retrieval using combination of semantic index and instance search. In *Advances in Multimedia Modeling*, pages 554–556. Springer, 2013.
- [50] Christopher Mueller, Martin Smole, and Klaus Schoeffmann. A demonstration of a hierarchical multi-layout 3d video browser. In *Multimedia and Expo Workshops (ICMEW), 2012 IEEE International Conference on*, pages 665–665. IEEE, 2012.

- [51] Klaus Schoeffmann, David Ahlstroem, Werner Bailer, Claudiu Cobarzan, Frank Hopfgartner, Kevin McGuinness, Cathal Gurrin, Christian Frisson, Duy-Dinh Le, Manfred Del Fabro, et al. The video browser showdown: a live evaluation of interactive video search tools. *International Journal of Multimedia Information Retrieval*, 3(2):113–127, 2014.
- [52] Ajay Divakaran, Clifton Forlines, Tom Lanning, Sam Shipman, and Kent Wittenburg. Augmenting fast-forward and rewind for personal digital video recorders. In *Consumer Electronics, 2005. ICCE. 2005 Digest of Technical Papers. International Conference on*, pages 43–44. IEEE, 2005.
- [53] Wolfgang Huerst and Philipp Jarvers. Interactive, dynamic video browsing with the zoomslider interface. In *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*, pages 4–pp. IEEE, 2005.
- [54] Klaus Schoeffmann. A user-centric media retrieval competition: The video browser showdown 2012-2014. *MultiMedia, IEEE*, 21(4):8–13, 2014.
- [55] Paul Over, Tzveta Ianeva, Wessel Kraaij, and Alan F Smeaton. Trecvid 2005 - an overview. 2005.
- [56] Paul Over, George M Awad, Jon Fiscus, Brian Antonishek, Martial Michel, Alan F Smeaton, Wessel Kraaij, and Georges Quenot. Trecvid 2010 - an overview of the goals, tasks, data, evaluation mechanisms, and metrics. 2011.
- [57] Usman Niaz, Miriam Redi, Claudiu Tanase, and Bernard Merialdo. Eurecom at trecvid 2014: The semantic indexing task. 2014.
- [58] Ben Shneiderman. *Designing the user interface: strategies for effective human-computer interaction*, volume 3. Addison-Wesley Reading, MA, 1992.
- [59] Jun Gong and Peter Tarasewich. Guidelines for handheld mobile device interface design. In *Proceedings of DSI 2004 Annual Meeting*, pages 3751–3756. Citeseer, 2004.
- [60] Hrvoje Benko, Andrew D Wilson, and Patrick Baudisch. Precise selection techniques for multi-touch screens. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 1263–1272. ACM, 2006.
- [61] Par-Anders Albinsson and Shumin Zhai. High precision touch screen interaction. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 105–112. ACM, 2003.

- [62] Shaun K Kane, Jeffrey P Bigham, and Jacob O Wobbrock. Slide rule: making mobile touch screens accessible to blind people using multi-touch interaction techniques. In *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility*, pages 73–80. ACM, 2008.
- [63] Uwe Laufs and Christopher Ruff. Aufbau von multi-touch-systemen. In *Multi-Touch*, pages 145–152. Springer, 2013.
- [64] Jochen Huber, Juergen Steimle, and Max Muehlhaeuser. Toward more efficient user interfaces for mobile video browsing: an in-depth exploration of the design space. In *Proceedings of the international conference on Multimedia*, pages 341–350. ACM, 2010.
- [65] VideoLAN Organization. Videolan vlc player for ios. <http://www.videolan.org/vlc/download-ios.html>. Eingesehen am 30.10.2015.
- [66] YouTube LLC. Youtube homepage. <https://www.youtube.com>. Eingesehen am 30.10.2015.
- [67] Github Inc. Github landing page. <https://github.com>. Eingesehen am 06.10.2015.
- [68] FFmpeg-Projekt. Ffmpeg landing page. <http://ffmpeg.org>. Eingesehen am 06.10.2015.
- [69] Scott Granneman, Clay Andres, Douglas Pundick, Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, et al. *Mac OS X Snow Leopard for Power Users*. Springer, 2010.
- [70] Didier Le Gall. Mpeg: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, 1991.
- [71] Apple Inc. Apple ios developer library: Mpmediaquery classreference. https://developer.apple.com/library/ios/documentation/MediaPlayer/Reference/MPMediaQuery_ClassReference/. Eingesehen am 08.10.2015.
- [72] Apple Inc. Apple ios developer library: Adding a segue between scenes in a storyboard. https://developer.apple.com/library/ios/recipes/xcode-help-IB_storyboard/Chapters/StoryboardSegue.html. Eingesehen am 08.10.2015.

- [73] Apple Inc. Apple ios developer library: Uiviewcontroller class. https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIViewController_Class/. Eingesehen am 08.10.2015.
- [74] Github alskipp. Asprogresspopupview repository. <https://github.com/alskipp/ASProgressPopUpView>. Eingesehen am 09.10.2015.
- [75] Apple Inc. Apple ios developer library: Nstimer class. https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSTimer_Class/. Eingesehen am 09.10.2015.
- [76] Dirk Louis and Peter Mueller. *Das Java 6 Codebook*. Pearson Deutschland GmbH, 2007.
- [77] Sandra G Hart and Lowell E Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. *Advances in psychology*, 52:139–183, 1988.
- [78] Bernard Rosner, Robert J Glynn, and Mei-Ling T Lee. The wilcoxon signed rank test for paired comparisons of clustered data. *Biometrics*, 62(1):185–192, 2006.

User ID: _____

Player ID: _____

Questionnaire



In this questionnaire will be asked questions about the previous used video player application.

This survey is for research purposes only and all information are anonymous. Please let us know, if you have any questions.

Please tick the box, which applies to you:

1. How mentally demanding was the task?

Very Low	Very High
----------	-----------

2. How physically demanding was the task?

Very Low	Very High
----------	-----------

3. How hurried or rushed was the pace of the task?

Very Low	Very High
----------	-----------

4. How successful were you in accomplishing what you were asked to do?

Perfect	Failure
---------	---------

5. How hard did you have to work to accomplish your level of performance?

Very Low	Very High
----------	-----------

6. How insecure, discouraged, irritated, stressed, and annoyed were you?

Very Low	Very High
----------	-----------

7. Do you wear glasses or contact lenses?

Yes No

8. Do you have a smart phone or tablet, or do you use these devices?

Yes No

If yes, since how many years? _____

User ID: _____

Player ID: _____

9. Age

_____ Years

10. Gender

female male

11. Rate the interaction of the previous used video player (how easy/intuitive). 1 is best, 6 is worst.

Rating: _____

12. Which player did you prefer and why?

13. Do you have suggestions to improve the tool?

Thanks for your participation!