

Visualisierungsinstitut der Universität Stuttgart (VISUS)

Universität Stuttgart
Allmandring 19
D–70569 Stuttgart

Masterarbeit

Streaming in web-based AR

Vinzenz Brantner

Studiengang: Softwaretechnik

Prüfer/in: Prof. Dr. Daniel Weiskopf

Betreuer/in: M. Sc. Seyda Öney,
M. Sc. Sergej Geringer
Dr. Jochen Schmid (ext. Trumpf)

Beginn am: 15. Mai 2023

Beendet am: 06. Dezember 2023

Danksagung

Ich möchte Seyda Öney und Sergej Geringer für ihre Betreuung und wertvolle Unterstützung herzlich danken. Ebenfalls danke ich Dr. Professor Daniel Weiskopf für die Annahme und Prüfung meiner Arbeit. Ein besonderer Dank geht an Dr. Jochen Schmid und sein Team für die Betreuung während meiner Zeit bei Trumpf Laser. Ich bin allen, die an diesem Projekt beteiligt waren, für ihre freundliche und hilfsbereite Unterstützung sehr dankbar.

Kurzfassung

Diese Arbeit stellt vor, wie ein Video-Stream mit niedriger Latenz im Webbrowser implementiert werden kann, um eine Echtzeit-Interaktion zu ermöglichen. Dabei werden verschiedene Streaming-Technologien wie Motion JPEG (MJPEG), Web Real-Time Communication (WebRTC), HLS (HTTP Live Streaming) und DASH (Dynamic Adaptive Streaming over HTTP) vorgestellt und evaluiert.

Des Weiteren wird aufgezeigt, wie der Video-Stream mit digitalen Informationen erweitert werden kann. Es wird dargelegt, wie 3D-Modelle im Browser augmentiert und entwickelt werden können.

Die Forschung erfolgte in Zusammenarbeit mit dem Unternehmen TRUMPF Laser, das Videostreaming für Steuerungs- und Dokumentationsaufgaben einsetzt. Ziel ist dabei die finale Darstellung im Browser, wobei der Video Stream eine möglichst geringe Latenz für Echtzeitübertragungen bieten und Bandbreiten effizient nutzen muss, ohne signifikante Qualitätseinbußen zu verzeichnen.

Im strukturellen Aufbau der Arbeit wird zunächst auf die Grundlagen von Browser, Video-Streaming und Augmentation eingegangen, gefolgt von einer Literaturrecherche zum aktuellen Stand der Technik. Danach werden die Implementierungsmöglichkeiten der Technologien und eine detaillierte Analyse der Streaming-Lösungen präsentiert. Zudem wird eine Designstudie für eine mögliche Exploration von potentiellen benutzerorientierten Visualisierungsoberflächen definiert. Für den Video Stream wird dann noch gezeigt, wie eine Augmentation von Elementen wie CAD-Overlays erweitert werden kann. Abschließend werden die Ergebnisse zusammengefasst und ein Ausblick auf zukünftige Entwicklungen im Bereich des Echtzeit-Streamings in Browsern gezeigt.

Inhaltsverzeichnis

1 Einleitung	17
1.1 TRUMPF Laser	18
1.2 Aufgabenstellung und struktureller Aufbau der Arbeit	19
2 Grundlagen	21
2.1 Webbrowser	21
2.2 JavaScript	25
2.3 WebAssembly	26
2.4 React.js	26
2.5 Python	27
2.6 Grundlagen der Bild- und Videokompression	27
2.7 Video-Streaming über HTTP	27
2.8 Augmented Reality (AR) und Visualisierung	28
3 Literaturrecherche	31
3.1 Bedeutung der Echtzeit-Latenz	31
3.2 Video-Streaming im Browser	32
3.3 Qualitätsmessung von Video-Streaming	34
3.4 Augmentation - Tracking & Mapping	37
4 Web-based Streaming	39
4.1 Anforderungen für das webbasierte Streaming	39
4.2 Beschreibung und Implementierung der Streaming-Lösungen	41
4.3 Entwicklung eines Streaming-Prototypen	48
4.4 Vergleich	51
5 Analyse von web-based streaming	55
5.1 Metriken	55
5.2 Methodik	55
5.3 Implementierung	57
5.4 Ergebnisse	59
5.5 Diskussion	64
6 Augmentation im web-based Streaming	67
6.1 Design Studie	67
6.2 Methodik	67
6.3 Interview & Ergebnisse	68
6.4 AR Grundlagen	70
6.5 AR Visualisierung	70
6.6 Diskussion	72

7 Zusammenfassung	75
7.1 Zusammenfassung und Schlussfolgerungen	75
7.2 Implikationen	76
7.3 Zukünftige Arbeit	76
Literaturverzeichnis	79

Abbildungsverzeichnis

2.1	Die Architektur des Chrome Browsers: Aufgeteilt in Prozesse und Threads [Goo18a].	22
2.2	Überblick über die verfügbaren Fetch Methoden auf verschiedenen Plattformen [Moz23b]	24
2.3	Überblick über Web-Kommunikationsprotokolle und ihre hierarchische Integration.	25
2.4	Exemplarische Darstellung einer Augmented-Reality-Anwendung, die mittels digitaler Informationen das Innere eines Objektes visualisiert [Vis23]	29
3.1	Aufbauten von videoLat Messungstool	35
3.2	Aufbauten von Low Delay Video Streaming	36
3.3	Modifizierter Aufbau	37
4.1	Darstellung einer Einzelbild-Video-Streaming-Architektur	43
4.2	Vereinfachter Beispielhafter Aufbau einer WebRTC Verbindung und deren Komponenten	46
4.3	Aufbau von HTTP Live Streaming (HLS) [App23]	48
4.4	Schematische Darstellung des Streaming-Prototyps	49
4.5	Setup eines aktiven Video Streams	50
5.1	Bildschirmaufnahme der Übertragung des QR Codes	57
5.2	Bildschirmaufnahme der Übertragung des QR Codes	58
5.3	Unterschiedliche Streaming Latenzen, keine Netzwerkbandbreite Einschränkung	61
5.4	Boxplot der Streaming-Latenzzeiten, bei hoher Auflösung, niedriger Netzwerkbandbreite	61
5.5	Boxplot der Streaming-Latenzzeiten (Neue Daten)	62
5.6	Vergleich der Bandbreiten Nutzung bei unterschiedlichen Einstellungen	63
5.7	Vergleich der Speicher und CPU Auslastung in Prozent	64
6.1	CAD 3D Overlay, welche mit realem Objekt übereinstimmt	72
6.2	CAD 3D Overlay mit OpenCV und Three.js	73

Tabellenverzeichnis

4.1 Vergleich verschiedener Streaming-Technologien anhand ausgewählter Kriterien.	53
5.1 Timestamp-Werte, die aus den Bildern extrahiert wurden	58

Verzeichnis der Listings

4.1	Beispiel Skript für Senden Einzelbilder über Http-Stream	42
4.2	Beispielskript zum Senden von Einzelbildern über WebSockets	44
6.1	Python-Code zur Echtzeit-Erkennung von ArUco-Markern in einem Videostream und zur Schätzung ihrer Pose mit OpenCV.	71

Abkürzungsverzeichnis

- AR** Augmented Reality. 28
- AV** Audio und Video. 47
- CDN** Content Delivery Network. 28
- DASH** Dynamic Adaptive Streaming over HTTP. 17
- FPS** Bilder pro Sekunde. 36
- HLS** HTTP Live Streaming. 9
- HTTP** Hypertext Transfer Protocol. 28
- JS** Javascript. 17
- MJPEG** Motion JPEG. 27
- VR** Virtual Reality. 28
- WASM** WebAssembly. 22
- WebRTC** Web Real-Time Communication. 41

1 Einleitung

Die Entwicklung von Benutzeroberflächen (User Interfaces) ist zunehmend durch den Einsatz von Browser-Technologien geprägt. Traditionelle Desktop-Anwendungen wie Excel oder Word, die ehemals ausschließlich als native Versionen verfügbar waren, werden nun durch Web-Versionen ergänzt. Außerdem erfolgt die Entwicklung neuer Applikationen häufig ausschließlich mit Webtechnologien, insbesondere im Bereich der Benutzeroberflächen, wie das Beispiel Microsoft Teams zeigt. Kommunikationsanwendungen, darunter Microsoft Teams¹ und Slack², nutzen Webtechnologien sowohl im Browser als auch in Desktop-Apps. Hierbei spielen Frameworks wie Electron³, die Javascript (JS), HTML und CSS verwenden, eine wichtige Rolle. Diese ermöglichen die Erstellung von "fast native"-Apps, die in Funktionalität Desktop-Anwendungen ähneln [off21]. Die zunehmende Verwendung dieser Technologien ist einerseits auf die Akzeptanz etablierter Sprachen wie JavaScript zurückzuführen und andererseits auf die Einführung von TypeScript⁴, einem von Microsoft unterstützten Open-Source-Projekt. TypeScript, bekannt für seine statische Typisierung, hat sich sowohl im Backend als auch in der Webentwicklung als eine der beliebtesten Sprachen etabliert. Die wachsende Bandbreite an Tools und Frameworks im JavaScript-TypeScript-Browser-Ökosystem erleichtert es Entwicklern zunehmend, interaktive und responsive Anwendungen zu erstellen [Sta23].

Allerdings sind Browser-basierte Anwendungen von standardisierten Browser-APIs abhängig, wodurch es in bestimmten Edge-Cases Einschränkungen gibt, da noch nicht alle Anwendungsfälle im Browser abgedeckt sind. Ein Beispiel hierfür ist das Echtzeit-Video-Streaming von einem Server zu einem Browser mit modernen Videocodec wie beispielsweise H.264. Zwar ist Video-Streaming und Live-Streaming (mit einer Latenz von über 2-3 Sekunden, HLS, Dynamic Adaptive Streaming over HTTP (DASH)) über den Browser üblich, jedoch stößt man auf Limitierungen, beispielsweise bei der Verarbeitung von Codecs und dem Datentransport [App23; SZM16].

Technologien im Bereich des Web-Streamings beinhalten verschiedene Techniken wie Motion JPEG (JPEG), Web Real-Time Communication (WebRTC) für Echtzeitkommunikation, sowie adaptive Streaming-Protokolle wie HTTP Live Streaming (HLS) und Dynamic Adaptive Streaming over HTTP (DASH). Diese Technologien ermöglichen eine Anpassung der Inhaltsübertragung an unterschiedliche Netzwerkbedingungen und besitzen jeweils spezifische Vor- und Nachteile.

¹Teams von Microsoft ermöglicht Chat- und Videokommunikation für effiziente Teamarbeit <https://www.microsoft.com/de-de/microsoft-365/get-started-with-teams-landing>.

²Slack ermöglicht die Kommunikation via Chat und Video sowie eine Dateifreigabe <https://slack.com/intl/de-de/what-is-slack>.

³Electron.js ist ein Open-Source-Framework zur Entwicklung von Desktop-Anwendungen mithilfe von Webtechnologien wie HTML, CSS und JavaScript.

⁴TypeScript ist eine Programmiersprache, die auf JavaScript aufbaut und statische Typisierung bietet.

1 Einleitung

Die Firma Trumpf Laser steht vor der Herausforderung, effektive Methoden für das Browser-basierte Streaming in Echtzeit zu identifizieren. Diese Masterarbeit zielt darauf ab, eine umfassende Analyse verschiedener Streaming-Lösungen durchzuführen. Im Fokus steht die Untersuchung der Eignung von Streaming-Technologien wie MJPEG, WebRTC, HLS und DASH für den Einsatz in Echtzeitanwendungen, insbesondere im industriellen Kontext. Zusätzlich soll die Implementierung eines Augmented Reality (AR) Szenarios im Browser erörtert werden.

Diese Arbeit befasst sich mit der Evaluierung von Werkzeugen in der Webentwicklung, insbesondere hinsichtlich Streaming-Technologien. Der Fokus liegt auf der Untersuchung, wie diese Technologien die Anforderungen an leistungsfähige und interaktive Nutzererfahrungen in der Industrie erfüllen. Ziel ist es, praxisorientierte Lösungen für Echtzeit-Übertragung und -Interaktion im Kontext von Webbrowsern zu erkunden. Ein praktisches Beispiel für die Anwendung solcher Technologien bietet das Unternehmen TRUMPF, das Videostreaming für Steuerung, Überwachung und Dokumentation in seiner Produktpalette einsetzt. Im Zuge einer kürzlich erfolgten Software-Reorganisation strebt TRUMPF danach, einen einheitlichen Unternehmensstil in den Benutzeroberflächen zu etablieren. Dies schließt die Entwicklung eigener UI-Komponenten ein, um eine konsistente Integration und Benutzerfreundlichkeit zu gewährleisten. Die zukünftige strategische Ausrichtung von TRUMPF wird immer stärker durch Webtechnologien bestimmt. Diese zeichnen sich durch ihre hohe Interaktivität und die vielfältigen Möglichkeiten moderner Browser-Plattformen aus. Die Technologien basieren auf Open-Source-Frameworks wie React.js. Ein markantes Beispiel für das Potenzial dieser Technologien ist der Einsatz von Chromium in einem Raumfahrzeug als Kontroll- und Informationsschnittstelle, was die Relevanz von Web-Streaming-Technologien unterstreicht [Inf22].

1.1 TRUMPF Laser

TRUMPF ist ein renommiertes High-Tech-Unternehmen, welches sich auf die Bereitstellung von Fertigungslösungen in den Domänen der Werkzeugmaschinen, Lasertechnologie und Elektronik spezialisiert hat [TRU23b]. Der Hauptsitz des Unternehmens befindet sich in Ditzingen, in der Nähe von Stuttgart, wo auch ein erheblicher Teil der Entwicklungsarbeit für CO2- und EUV-Laser stattfindet. Diese Laser-Technologien sind von entscheidender Bedeutung für die globale Halbleiterindustrie der kommenden Jahrzehnte [TRU23c].

Mit über 1500 Mitarbeitern, darunter 300 in der Entwicklung, ist Schramberg ein wichtiger Standort von TRUMPF. Es ist der Hauptstandort für die Entwicklung und Herstellung von Festkörperlasern, die in der Elektromobilität und Solarindustrie als essenzielle Werkzeuge in der Produktion dienen [TRU23b].

Diese Arbeit wird in Kooperation mit der Firma TRUMPF Laser erarbeitet, unter anderem innerhalb des Teams, welches für das Produkt VisionLine von TRUMPF verantwortlich ist. VisionLine, ein fortschrittliches Bildverarbeitungssystem von TRUMPF, zielt darauf ab, fehlerhafte Teile in Schneid- und Schweißprozessen zu identifizieren und zu eliminieren. Dank der kamerabasierten Technologie erkennt VisionLine die Position von Bauteilen automatisch, kommuniziert diese Informationen an das Steuerungssystem und sorgt so für die korrekte Platzierung der Schweißnähte [TRU23a]. In Bezug auf Videostreaming und die Darstellung virtueller Informationen, wie beispielsweise CAD-Modelle, ist es wesentlich, dass die finale Darstellung im Browser stattfindet. Dabei sollte die Videoübertragung eine sehr geringe Latenz für Echtzeit-Übertragungen bieten. Es ist auch

wichtig, Rechenbeschränkungen zu berücksichtigen, Lösungen für die verfügbaren Ressourcen zu optimieren, die Bandbreite effizient zu nutzen ohne erheblichen Qualitätsverlust und die Möglichkeit zu bieten, die Videoübertragung auf mehreren Clients gleichzeitig anzeigen zu lassen.

1.2 Aufgabenstellung und struktureller Aufbau der Arbeit

Im Kapitel (2) werden die grundlegenden Aspekte, die für Streaming und Augmentation relevant sind, dargelegt. Dies beinhaltet die Funktionsweise von Webbrowsern, Grundlagen über Programmiersprachen, sowie die Übertragung von Video- und Bilddaten und Augmentations-Techniken.

Das Kapitel (3) umfasst eine umfassende Literaturrecherche zum aktuellen Stand der Technik in Bezug auf Echtzeit-Videoübertragung in Browern. Besonderes Augenmerk liegt auf der Untersuchung von niedriger Latenz und der Bewertungsmöglichkeiten von Videostreams hinsichtlich ihrer Qualität, wie etwa Latenz. Ebenfalls behandelt wird die Definition und Klassifizierung von 'Echtzeit' im Kontext von Latenzparametern.

Im Kapitel (4) werden die Streaming-Technologien und deren Implementierungsmöglichkeiten detailliert betrachtet. Es wird erörtert, wie das Streaming auf verschiedenen Geräten und unter unterschiedlichen Hardware-Konfigurationen realisiert werden kann. Basierend auf den Erkenntnissen aus der Literaturrecherche und technologischen Grundlagen, wird eine webbasierte Streaming-Lösung entwickelt.

Das Kapitel (5) konzentriert sich auf die Analyse der implementierten Streaming-Technologien. Untersucht werden Metriken wie Latenz, CPU-Auslastung, Speicherbedarf und Bandbreite. Die Methodik zur Messung und zum Vergleich der Streaming-Lösungen wird hierbei besonders hervorgehoben.

Im Kapitel (6) wird das Konzept einer Designstudie im Bereich der Visualisierung definiert. Anhand eines Interviews mit einem Domänenexperten werden potentielle Daten für die Visualisierung identifiziert, um dem Nutzer einen Mehrwert zu bieten. Ziel ist es, eine Benutzeroberfläche zu entwickeln, die den Videostream um zusätzliche Elemente erweitert und Interaktionen mit dem gestreamten Video sowie den zusätzlichen Informationen, wie CAD-Overlays, ermöglicht.

Abschließend in Kapitel 7 werden in diesem Forschungsprojekt die Ergebnisse zusammengefasst und ein Ausblick auf zukünftige Technologien gegeben. Das Ziel ist es, eine robuste und nutzerorientierte Lösung für Echtzeit-Streaming in Browern zu entwickeln und gleichzeitig neue Wege in der Informationsintegration und -darstellung zu erschließen.

2 Grundlagen

Dieses Kapitel beschäftigt sich mit den grundlegenden Technologien und Konzepten, die in dieser Arbeit Verwendung finden. Im Detail werden folgende Komponenten erörtert: Webbrowser, JavaScript, das JavaScript UI Framework React.js, die Programmiersprache Python, Bildkodierung (Image Encoding), Videotechnologien sowie Augmented Reality (AR).

2.1 Webbrowser

Diese Arbeit setzt Webbrowser als zentrale Plattform voraus. Im Folgenden wird die interne Funktionsweise von Webbrowsern erläutert und bestehende Limitationen werden beleuchtet. Ein Webbrowser ist eine Softwareapplikation, die für die Abfrage und Darstellung von Dokumenten von einem Remote-Server oder aus dem World Wide Web (WWW) konzipiert ist [BF11]. Das WWW, entwickelt von Tim Berners-Lee, ermöglicht den Zugriff auf Informationen über ein Netzwerk und wird durch Uniform Resource Identifiers (URIs) identifiziert [BF11]. Das initiale Dokument ist meist in HyperText Markup Language (HTML) verfasst, einem fundamentalen Baustein des Internets. Die Datenübertragung zwischen Browser und Server erfolgt über das Hypertext Transfer Protocol (HTTP) [GG06]. HTML dient zur Strukturierung und visuellen Gestaltung von Webinhalten. In Kombination mit Cascading Style Sheets (CSS) und JavaScript (JS) ermöglicht HTML dem Browser die Ausführung dynamischer Webanwendungen [MDN23; TLV08]. Webbrowser unterstützen neben HTML auch die Darstellung von Bildern, die Wiedergabe von Videos und das Anzeigen von PDFs. Sie weisen allerdings im Vergleich zu nativen Anwendungen gewisse Einschränkungen auf. Während native Anwendungen den vollen Funktionsumfang eines Betriebssystems nutzen können, müssen Webanwendungen auf die APIs des Browsers zurückgreifen [Ama23]. Webbrowser gehören zu den am häufigsten genutzten Anwendungen auf Computern und finden vielfältige Verwendung, beispielsweise beim Lesen von Nachrichten, in Unterhaltungsmedien, in sozialen Netzwerken, bei E-Mails und sogar in Raumfahrtmissionen [Inf22].

Basierend auf der Forschung von Grosskurth und Godfrey [GG06] und den von Google LLC veröffentlichten Blog Beiträgen wird im folgenden die allgemeine Architektur eines Browser abgeleitet. Die Architektur, die Grosskurth und Godfrey [GG06] durch die Untersuchung des Quellcodes von Mozilla Firefox ableiten, zeigt Ähnlichkeiten mit der von Google beschriebenen Architektur für Chrome [Goo18a; Goo18b; Goo18c]. In Blogbeiträgen von Google wird Chrome in oder basierend auf Prozesse bzw. Threads dargestellt, die Parallelen zu den High-Level-Bausteinen im Paper von Grosskurth aufweisen. Abbildung 2.1 visualisiert diese architektonischen Komponenten und ihre Verbindungen.

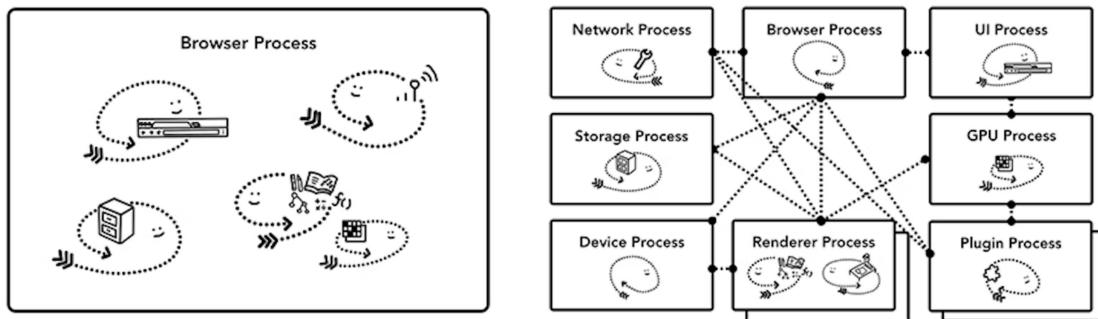


Abbildung 2.1: Die Architektur des Chrome Browsers: Aufgeteilt in Prozesse und Threads [Goo18a].

2.1.1 High Level Architecture

User Interface

Dieser Bereich repräsentiert die Benutzeroberfläche eines Browsers, einschließlich der Darstellung des HTML-Dokuments, Navigationselementen und weiteren benutzerfreundlichen Funktionen [GG06].

Browsing Engine

Die Browsing Engine dient als Schnittstelle zwischen der Benutzeroberfläche und anderen Prozessen, wie etwa der Rendering Engine. Sie steuert die Aktivitäten in der Adressleiste und handhabt Netzwerkanfragen [GG06].

Rendering Engine

Die Rendering Engine ist in der Lage, HTML, CSS und JavaScript zu interpretieren und in eine interaktive Webseite umzuwandeln. Sie konvertiert das HTML-Dokument in ein Document Object Model (DOM), welches JavaScript-Interaktionen und dynamische Anpassungen ermöglicht [GG06; Goo18b]. Neben JavaScript unterstützt die Rendering Engine auch WebAssembly (WASM), das ein schnelleres Ausführen von Code im Browser erlaubt, sowie auch ermöglicht, andere Sprachen welche sich zu WASM compilieren lassen, auszuführen. Zum Beispiel Programmiersprachen wie C++ oder Rust können zu WASM kompiliert werden. Jedoch hat auch WASM Einschränkungen, wie z.B. das Fehlen einer automatischen Speicherbereinigung (Garbage Collection) und keinen direkten Zugriff auf DOM-Elemente [Moz23a].

Netzwerk Block

Dieser Abschnitt ist verantwortlich für die Datenübertragung zwischen Server und Browser. Er verwaltet Anfragen, Antworten und andere sicherheitsrelevante Aspekte der Netzwerkkommunikation. Ein zentrales Kommunikationsprotokoll, das Hypertext Transfer Protocol (HTTP), wird hier implementiert [Chr23; Goo18c]. HTTP ist menschenlesbar und ermöglicht verschiedene Methoden für spezifische Aktionen sowie unterschiedliche Inhaltsarten [Clo23].

Speicher Block

Der Speicher Block befasst sich mit allen Aufgaben der persistenten Datenspeicherung, einschließlich des Cachings von häufig genutzten Ressourcen. Cookies und andere Speicheroptionen ermöglichen es Webentwicklern, benutzerspezifische Daten zu speichern [GG06]. In modernen Webanwendungen spielen Cookies eine essenzielle Rolle für die Speicherung von benutzerspezifischen Informationen. Sie bewahren Sitzungsinformationen, Benutzereinstellungen und andere relevante Daten zwischen den Browsersitzungen. Dies verbessert die Benutzererfahrung durch das Beibehalten von Präferenzen und Anmeldeinformationen und ermöglicht personalisierte Inhalte und Werbung. Zusätzlich zu Cookies bieten moderne Browser Speichermöglichkeiten wie Local Storage, Session Storage und IndexedDB, die Entwicklern mehr Flexibilität und Kontrolle über die Datenspeicherung geben.

UI Block

Der UI-Block stellt die grafische Benutzeroberfläche dar, die dem Nutzer im Webbrowser präsentiert wird. Dieser ist sowohl mit der Grafikverarbeitungseinheit (GPU) als auch mit dem eigentlichen Browserprozess verbunden. Diese Anbindung ermöglicht eine effiziente Darstellung und Interaktion innerhalb des Browsers. Der UI-Block umfasst wichtige Elemente wie Scrollleisten, Symbolleisten, Add-ons, Einstellungen und Druckfunktionen [GG06; Goo18c].

2.1.2 Web-APIs (Browser-APIs)

Web-APIs (Application Programming Interfaces) sind für die Programmierung und Ausführung von Anwendungen im Webbrowser essenziell. Sie bieten Zugang zu einer Vielzahl von Computertechnologien und sind in der Regel über JavaScript oder WebAssembly zugänglich [Moz23f]. Browserhersteller implementieren diese Schnittstellen gemäß standardisierten Spezifikationen, wodurch Funktionen wie beispielsweise die Audiowiedergabe über Computerlautsprecher ermöglicht werden. Obgleich diese APIs für JS verfügbar sind, werden sie oft in leistungsfähigeren, niedrigstufigen Programmiersprachen wie C++ oder Rust implementiert [Moz23d].

Ein essenzieller Aspekt in der Webentwicklung ist die Tatsache, dass nicht alle in einem bestimmten Browser verfügbaren Web-APIs zwangsläufig in anderen Browsern vorhanden sind. Dies stellt Entwicklerinnen und Entwickler vor die Herausforderung, ihre Anwendungen individuell an unterschiedliche Browser anzupassen. Ein exemplarisches Beispiel hierfür ist die Implementierung

2 Grundlagen

	□														≡		
	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android	Deno	Node.js				
<code>fetch</code>	✓ 42	✓ 14	✓ 39	✓ 29	✓ 10.1	✓ 42	✓ 39	✓ 29	✓ 10.3	✓ 4.0	✓ 42	✓ 1.0	✓ 18.0.0	*			
Authorization header removed from cross-origin redirects	✗ No	✗ No	✓ 111	✗ No	✓ 16.1	✗ No	✓ 111	✗ No	✓ 16.1	✗ No	✗ No	✗ No	✗ ?				
Support for blob: and data:	✓ 48	✓ 79	✓ 39	✓ 35	✓ 10.1	✓ 48	✓ 39	✓ 35	✓ 10.3	✓ 5.0	✓ 43	✓ 1.9	✓ ?				
<code>init.keepalive</code> parameter	✓ 66	✓ 15	✗ No	✓ 53	✓ 13	✓ 66	✗ No	✓ 47	✓ 13	✓ 9.0	✓ 66	✗ No	✓ ?				
<code>init.priority</code> parameter	✓ 101	✓ 101	✗ No	✓ 87	✓ No	✓ 101	✗ No	✓ 70	✗ No	✓ 19.0	✓ 101	✗ No	✓ ?				
<code>init.referrerPolicy</code> parameter	✓ 52	✓ 79	✓ 52	✓ 39	✓ 11.1	✓ 52	✓ 52	✓ 41	✗ No	✓ 6.0	✓ 52	✗ No	✓ ?				
<code>init.signal</code> parameter	✓ 66	✓ 16	✓ 57	✓ 53	✓ 11.1	✓ 66	✓ 57	✓ 47	✓ 11.3	✓ 9.0	✓ 66	✓ 11.1	✓ 18.0.0				
Available in workers	✓ 42	✓ 14	✓ 39	✓ 29	✓ 10.1	✓ 42	✓ 39	✓ 29	✓ 10.3	✓ 4.0	✓ 42	✓ Yes	✓ ?				

Abbildung 2.2: Überblick über die verfügbaren Fetch Methoden auf verschiedenen Plattformen [Moz23b]

der Fetch-Methode, die häufig für den REST-basierten¹ Datenaustausch mit Servern genutzt wird. Die Realisierung dieser Methode variiert in diversen Browsern wie Chrome oder Firefox, wie in Abbildung 2.2 veranschaulicht wird. Ein spezifisches Merkmal, wie das Entfernen des "Authorization Headers" bei Cross-Origin-Weiterleitungen, ist beispielsweise in Firefox² und Safari³, jedoch nicht in Chrome⁴ implementiert [Moz23b].

In diesem Kontext nimmt das World Wide Web Consortium (W3C) eine Schlüsselposition ein. Das W3C, eine Koalition verschiedener Interessengruppen, engagiert sich in der Entwicklung und Spezifikation von Web-APIs. Es zielt darauf ab, durch die Festlegung von Standards in der Webentwicklung und -technologien die Interoperabilität zu verbessern und das kontinuierliche Wachstum des Internets zu unterstützen. Diese Standards tragen entscheidend dazu bei, die einheitliche Implementierung von Web-APIs über verschiedene Browser hinweg zu erleichtern und somit die Kompatibilität und Benutzerfreundlichkeit zu erhöhen [Wor23].

¹REST (Representational State Transfer) ist ein Architekturstil für die Kommunikation zwischen verteilten Systemen und ermöglicht einen strukturierten und ressourcenorientierten Datenaustausch zwischen Servern und Clients in Webanwendungen.

²Mozilla Firefox ist ein freier Webbrowser, entwickelt von der Mozilla Foundation. <https://www.mozilla.org/firefox/>

³Safari, entwickelt von Apple Inc., ist ein Webbrowser für macOS und iOS. <https://www.apple.com/safari/>

⁴Google Chrome ist ein Webbrowser von Google LLC, verfügbar für verschiedene Betriebssysteme. <https://www.google.com/chrome/>

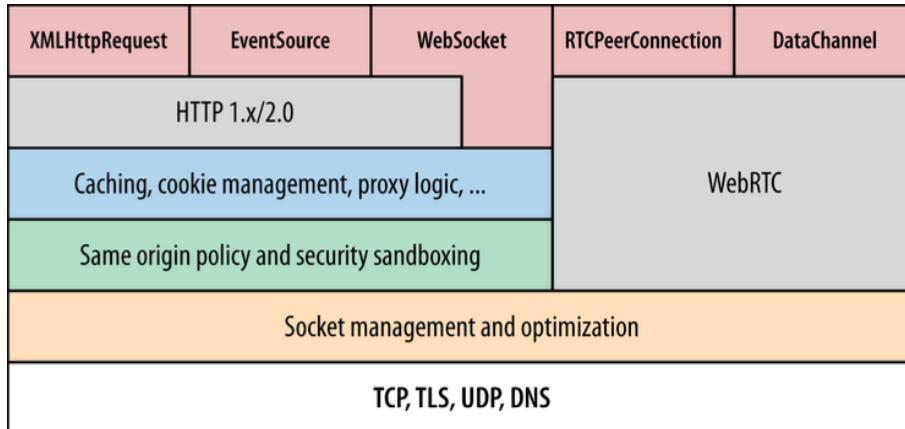


Abbildung 2.3: Überblick über Web-Kommunikationsprotokolle und ihre hierarchische Integration.

2.1.3 Einschränkungen des Browsers

Wie zuvor in Abschnitt 2.1.2 erörtert, legen Web-APIs die Funktionsweise einer Webanwendung fest, insbesondere in Bezug auf die Interaktion mit spezifischen Elementen des zugrundeliegenden Betriebssystems. Die Fetch-API beispielsweise bietet umfangreiche Funktionen für HTTP-Anfragen, um beispielsweise Daten in Javascript Anwendungen von externen Servern zu holen. Allerdings illustriert Abbildung 2.2, dass nicht jede Methode der Fetch-API in allen Browsern verfügbar ist.

Einschränkungen treten auch im Bereich des Echtzeit-Streamings auf, besonders hinsichtlich des Datentransports und der Dekodierung von Mediendaten. Idealerweise sollte der Datentransport minimale Overhead-Kosten verursachen, während die Dekodierung möglichst effizient erfolgen muss. Die Konstruktionsprinzipien moderner Browser entbinden Entwickler von der Notwendigkeit, Netzwerk-Sockets eigenhändig zu verwalten. Stattdessen stellen Web-APIs abstrahierte Schnittstellen zu verschiedenen Netzwerkelementen bereit [Gri13]. Wie in Abbildung 2.3 dargestellt, bieten Browser Methoden für die grundlegende Ebene des Datentransports, wie WebSockets und Event-Sourcen, an. Dies begrenzt allerdings die Möglichkeit, ein eigenes, hochgradig optimiertes Socket-Management zu implementieren. In der Praxis sind Anwendungen im Browser auf die Kommunikation über die vorhandenen Web-APIs beschränkt. Für die Medien-Dekodierung wäre ein direkter Hardwarezugriff, z.B. auf die GPU, vorteilhaft. Hier könnte die zukünftige WebCodecs-API einen effizienteren Weg zur Dekodierung von Medien im Browser bieten [Moz23g].

2.2 JavaScript

JavaScript ist einer der wichtigsten Bestandteile des Browsers. Neben WASM, welches im Abschnitt 2.3 erwähnt wird, ist es die einzige Programmiersprache, die der Browser ausführen kann. JavaScript ist eine just-in-time kompilierte Programmiersprache und wird als „high-level“ betrachtet. Mit „high-level“ ist gemeint, dass der Entwickler nicht mit der gesamten Komplexität konfrontiert ist, wie es bei Sprachen der Fall ist, die näher am Maschinencode sind, beispielsweise die Sprache C. Maschinencode ist die Sprache, die der Computer letztendlich ausführen kann. JavaScript wurde

2 Grundlagen

1995 mit Netscape Navigator veröffentlicht und 1997 unter dem Namen ECMAScript standardisiert. Inzwischen gibt es die ECMA Spezifikation ECMAScript2024 [Ecm23], und sowohl JavaScript als auch seine typisierte Variante TypeScript gehören zu den beliebtesten Programmiersprachen.

JavaScript hat sich nicht nur durch seine Vielseitigkeit und Anpassungsfähigkeit ausgezeichnet, sondern auch durch seine Community und die Vielzahl an verfügbaren Bibliotheken und Frameworks. Diese Faktoren haben dazu beigetragen, dass JavaScript eine unverzichtbare Rolle in der Webentwicklung spielt und eine breite Palette von Anwendungsfällen abdeckt, von einfachen Websites bis hin zu komplexen Webanwendungen und sogar serverseitigen Anwendungen mit Node.js⁵.

2.3 WebAssembly

WASM ist ein binäres Befehlsformat, das als portables Kompilationsziel für verschiedene Programmiersprachen dient. Dies ermöglicht eine effiziente und schnelle Bereitstellung im Web für Client- und Serveranwendungen. WASM arbeitet in einer speichersicheren, abgekapselten Ausführungsumgebung, die in bestehende JavaScript-Virtualmaschinen integriert werden kann und dabei den Sicherheitsrichtlinien von Browsern entspricht. Es ist so strukturiert, dass es mit nahezu nativer Geschwindigkeit durch die Nutzung allgemeiner Hardwarefähigkeiten auf unterschiedlichen Plattformen ausgeführt wird. WASM Module können dann auch über JavaScript ausgeführt werden. WASM selbst kann über Web-APIs auf Browserfunktionen zugreifen [W3C23]

2.4 React.js

React.js⁶ ist eine JavaScript-Bibliothek zur Entwicklung von Benutzeroberflächen. Sie zeichnet sich durch APIs aus, die das deklarative Erstellen interaktiver User Interfaces für verschiedene Zustände einer Applikation ermöglichen. React fördert ein Entwicklungsmodell, bei dem User Interface-Teile als separate Komponenten entwickelt werden. Diese Komponenten können kombiniert werden, um komplexe Interfaces zu erstellen. Neben der Web-Version bietet React auch die Möglichkeit, mit React Native native Apps zu entwickeln. React Native erweitert die Funktionalitäten von React, indem es die Erstellung von nativen Applikationen für mobile Plattformen ermöglicht.⁷.

In den letzten Jahren hat sich React.js als das populärste Tool für die Entwicklung von UI-Frameworks etabliert. Eine Umfrage zeigt, dass 50% der Entwickler, die im Bereich der Webtechnologien tätig sind, React.js verwendet haben. Dies bezeugt nicht nur die Popularität der Bibliothek, sondern auch das Vertrauen, das Entwickler in sie setzen. Durch kontinuierliche Weiterentwicklung und Verbesserungen sichert React.js seine zentrale Stellung in der Frontend-Entwicklung auch für die Zukunft [Sta23].

⁵Node.js ist eine JavaScript-Laufzeitumgebung, die auf der V8 JavaScript Engine basiert und für die Entwicklung skalierbarer Netzwerkanwendungen verwendet wird. <https://nodejs.org/>

⁶React ist eine JavaScript-Bibliothek für den Bau von Benutzeroberflächen, entwickelt von Facebook. <https://reactjs.org/>

⁷React.js: Eine JavaScript-Bibliothek zur Erstellung von Benutzeroberflächen, die es ermöglicht, interaktive UIs effizient und deklarativ zu erstellen. <https://react.dev/>

2.5 Python

Python ist wie JavaScript eine vielseitig einsetzbare, hochstufige Programmiersprache mit einem dynamischen Typsystem und einer einfachen Syntax [Sev15]. In den letzten Jahren hat Python, insbesondere durch das Full-Stack-Webentwicklungsframework Django sowie in den Bereichen Datenanalyse und maschinelles Lernen, stark an Popularität gewonnen. Laut der Umfrage von Stack Overflow [Sta23] zählt Python, neben JavaScript, zu den am häufigsten genutzten Programmiersprachen. Die Beliebtheit von Python ist unter anderem auf die große und aktive Community sowie die umfangreiche Verfügbarkeit von Werkzeugen und Bibliotheken zurückzuführen. Diese Ressourcenvielfalt ermöglicht es Entwicklern, ein breites Spektrum von Anwendungen effizient zu entwickeln und umzusetzen.

2.6 Grundlagen der Bild- und Videokompression

Die digitale Bildverarbeitung umfasst mehrere Schritte, beginnend mit der Bildaufnahme, gefolgt von verschiedenen Bildverarbeitungstechniken wie Filterung, Verbesserung und Farbkorrektur. Ein wesentlicher Abschluss dieses Prozesses ist die Kompression oder Kodierung der Bilder [GW18, S. 41f.]. Bild- und Videokompression sind entscheidend für eine effiziente Datenübertragung und haben das Ziel, die Datenmenge zu reduzieren, ohne die Qualität und Integrität der Inhalte wesentlich zu beeinträchtigen.

Ein zentraler Aspekt der Kompression ist die Reduzierung von Kodierungsredundanz. Hierbei werden häufig vorkommende Symbole oder Bildteile identifiziert und effizienter kodiert. Ebenso relevant sind die Konzepte der räumlichen und zeitlichen Redundanz sowie das Entfernen irrelevanter Informationen. Räumliche Redundanz bezieht sich auf die Ähnlichkeiten zwischen benachbarten Pixeln in einem Bild, während zeitliche Redundanz in der Videokompression genutzt wird, um die Ähnlichkeiten zwischen aufeinanderfolgenden Frames auszunutzen [GW18, S. 540]. Ein prominentes Beispiel für ein verlustfreies Kompressionsverfahren ist die Huffman-Codierung. Diese Technik ordnet häufig vorkommenden Symbolen kürzere Binärcodes zu. Dies geschieht auf Basis einer Frequenzanalyse und dem Aufbau eines Huffman-Baumes, was eine effiziente Reduzierung der Datenmenge ohne Informationsverlust ermöglicht [GW18, S. 553f]. Die Herausforderung in der Bild- und Videokompression liegt darin ein optimum zwischen Effizienz und Datenqualität zu finden. Es gilt, die Datenmenge soweit wie möglich zu reduzieren, ohne dabei die visuelle Qualität für den Betrachter spürbar zu beeinträchtigen.

2.7 Video-Streaming über HTTP

Die behandelt das Thema Video-Streaming im Web, unter Berücksichtigung verschiedener Formate und Kodierungsmethoden. Ein zentraler Ansatz hierbei ist Motion JPEG (MJPEG), welches auf der verlustbehafteten Intraframe-Kodierung basiert und primär für die zeitliche Abfolge von Standbildern eingesetzt wird [Bin15, S. 66f]. Es ist anzumerken, dass MJPEG keine zeitliche Redundanz verwendet, im Gegensatz zu neueren Videokodierungsstandards wie H.265/HEVC.

2 Grundlagen

Im Kontrast hierzu stehen Formate wie HLS und DASH, die auf MPEG-Codecs aufbauen und die Möglichkeit der Interframe-Kodierung bieten [Bin15, S. 257, S.297]. Diese Techniken sind insbesondere bei Videos mit geringer bis mittlerer Bewegung effektiv, da sie zeitliche Redundanzen zwischen aufeinanderfolgenden Frames ausnutzen [Bin15, S. 66f].

Ein weiterer wichtiger Aspekt ist die Übertragung von Streaming-Inhalten über Hypertext Transfer Protocol (HTTP), die in Webbrowsersn dargestellt werden können. Ein Hauptgrund für die Beliebtheit des Video-Streamings über HTTP liegt in der Fähigkeit dieses Protokolls, die zuverlässige Übertragung von Netzwerkpaketen zu gewährleisten. Dies ist besonders bei hochkomprimierten Videos relevant, die anfällig für Datenverlust sind. Zudem arbeitet HTTP effizient mit bestehenden Infrastrukturen wie Content Delivery Network (CDN)s, Caches und anderen netzwerkbasierten Infrastrukturen zusammen [Bin15, S. 246f]. Bei MJPEG kann jedes Frame einzeln über HTTP übertragen werden, während bei HLS und DASH Playlists und Fragmente zum Einsatz kommen. Diese Fragmente werden mittels HTTP heruntergeladen und die Downloads über die Playlists koordiniert [Bin15, S. 247].

Zusammenfassend bietet sowohl MJPEG als auch HLS und DASH spezifische Vorteile in ihren jeweiligen Anwendungsbereichen. Während sich MJPEG besonders für Videos mit hoher Bewegung eignet, bieten HLS und DASH mit ihrer Interframe-Kodierung eine effizientere Kompression bei geringer bis moderater Bewegung. Diese Vielfalt spiegelt die breiten Anforderungen und Möglichkeiten im Bereich des Web-Video-Streamings wider.

2.8 Augmented Reality (AR) und Visualisierung

Augmented Reality (AR), auch als erweiterte Realität bekannt, ist eine Technologie, die es ermöglicht, Informationen und virtuelle Objekte nahtlos in die reale Umgebung des Benutzers zu integrieren. Mithilfe von Digitalgeräten mit einem Bildschirm oder AR-Brillen werden digitale Inhalte über die wahrgenommene Umwelt gelegt, um so eine erweiterte Wirklichkeit zu schaffen. Die Anwender können somit in Echtzeit interaktive Elemente sehen und verwenden, während sie den Kontakt zur tatsächlichen Welt behalten. Diese Technik bietet im Unterschied zur Virtual Reality (VR), die einen vollständigen Ersatz der Wirklichkeit darstellt, eine Verschmelzung von realen und virtuellen Ansichten [Azu97; NO23, S. 24f.]. Die Abbildung 2.4 zeigt eine beispielhafte Anwendung. Hier wird veranschaulicht, wie man das Innere eines verschlossenen realen Objekts durch digitale Informationen sichtbar machen kann.

2.8 Augmented Reality (AR) und Visualisierung

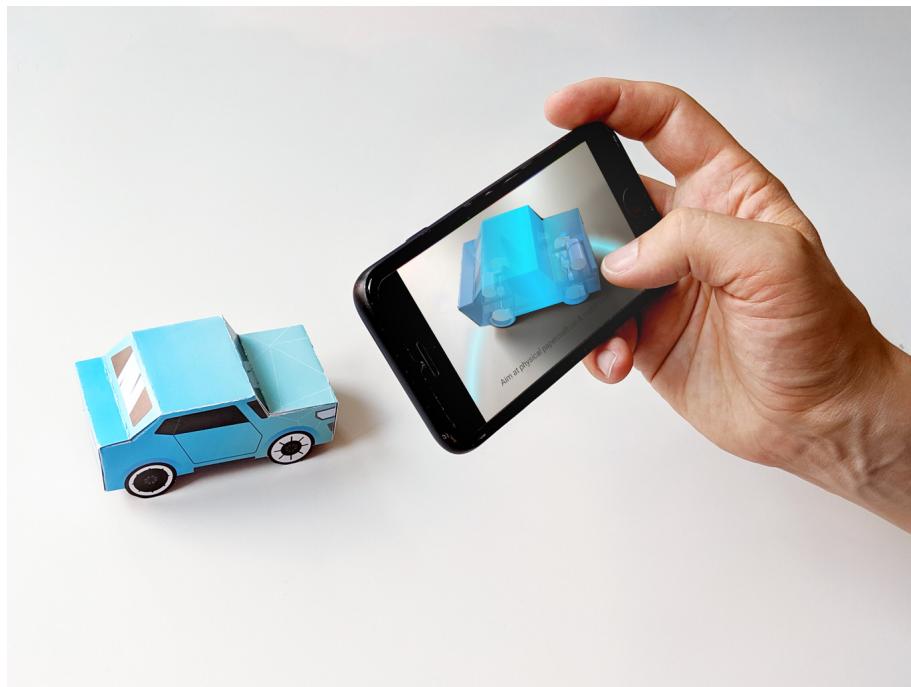


Abbildung 2.4: Exemplarische Darstellung einer Augmented-Reality-Anwendung, die mittels digitaler Informationen das Innere eines Objektes visualisiert [Vis23]

3 Literaturrecherche

In diesem Kapitel wird ein Überblick über die aktuelle Literatur zu verschiedenen Aspekten der Videostreaming-Technologie gegeben. Es beginnt mit einer Übersicht darüber, was 'Realität im Kontext von Videostreaming' bedeutet. Anschließend wird das Thema des Videostreamings mit geringer Latenz behandelt, welche Technologien es dazu gibt. Des Weiteren wird das Thema behandelt, wie die Qualität von Videostreams gemessen werden kann. Zusätzlich werden wichtige Aspekte wie Augmentierung und Registrierung im Videostreaming erläutert.

3.1 Bedeutung der Echtzeit-Latenz

Latenz ist entscheidend im Bereich des Video-Streamings, besonders bei Echtzeit-Interaktionen, wie zum Beispiel Entscheidungen auf Basis der Videodaten. Wie in Abschnitt 1 erwähnt, ist eines der Hauptziele herauszufinden, wie ein Video Stream im Browser dargestellt werden kann, um die Beobachtung und Steuerung von ferngesteuerten Maschinen zu ermöglichen. Daher ist es entscheidend, die akzeptablen Latenzgrenzen zu erläutern, die eine effektive Maschinensteuerung gewährleisten.

Das Konzept der "wahrgenommenen Latenz" ist für das Verständnis der Mensch-Maschine-Interaktion im Kontext des Video-Streamings wichtig. Einfach ausgedrückt bezieht sich die wahrgenommene Latenz auf die spürbare Verzögerung, die ein Endbenutzer beim Interagieren mit einem System erkennen kann. Diese Verzögerung kann sich in verschiedenen Formen manifestieren, wie zum Beispiel einer merklichen Verzögerung bei der Einleitung einer Aktion oder einem spürbaren Hindernis, das die Effizienz oder das Ergebnis einer Aufgabe negativ beeinflussen könnte. Beispielsweise könnte ein Benutzer einen wahrnehmbaren Unterschied in den Reaktionszeiten beim Manövrieren einer Drohne oder bei der Ausführung präziser Operationen an einer entfernt gelegenen Maschine erkennen. Dieser Begriff der wahrnehmbaren Latenz ist bedeutend, da er direkt die Benutzererfahrung beeinflusst, was wiederum kaskadierende Auswirkungen auf das Ergebnis der Aufgabe oder die breiteren Ziele des Benutzers haben kann [Bac19; ETS23].

Aus der vorhandenen Literatur lassen sich Erkenntnisse über die akzeptablen Latenzschwellen für eine Reihe von Anwendungen gewinnen. Zum Beispiel:

- **Gaming und GUI-Interaktionen:** In Gaming-Szenarien, insbesondere solchen, die Echtzeitstrategien oder schnelle Manöver beinhalten, wird eine Latenz von etwa 50 ms benötigt. Ähnlich fallen auch interaktive grafische Benutzeroberflächen in diesen Latenzbereich, da jegliche Verzögerungen eine nahtlose Navigation oder Interaktion behindern könnten [Bac19].

- **Kopfmontierte Displays:** In Kontexten, in denen Benutzer auf kopfmontierte Displays angewiesen sind, werden die Latenzanforderungen noch strenger und verlangen oft, dass die Reaktionszeiten auf 15 ms begrenzt werden. Dies ist wesentlich, um sicherzustellen, dass das visuelle Feedback nahtlos mit den physischen Bewegungen des Benutzers übereinstimmt und so Desorientierung oder Übelkeit verringert wird [Bac19].
- **AR-basierte Fernsteuerung für selbstfahrende Fahrzeuge:** Wenn man den Umfang auf komplexere Anwendungen wie AR-basierte Steuerungssysteme für autonome Fahrzeuge erweitert behaupten El Marai et al. [ETS23] die Ansicht, dass für solche Szenarien akzeptable Latenzen bis zu 800 ms betragen können.

Latenz, im Kontext des Video-Streamings für die Fernsteuerung von Maschinen, tritt als ein wesentlicher Parameter hervor, der sowohl die Benutzererfahrung als auch die Wirksamkeit der Aufgabenausführung beeinflusst. Obwohl die Benchmarks je nach Anwendung variieren, ist es von größter Wichtigkeit, die Latenzanforderungen an die spezifische Art und die Anforderungen der jeweiligen Aufgabe anzupassen. Mit der kontinuierlichen Weiterentwicklung der Technologie wird es zwingend erforderlich sein, diese Benchmarks kontinuierlich zu überprüfen und neu zu definieren, um sie an die sich entwickelnden Bedürfnisse der Benutzer und technologischen Fortschritte anzupassen.

Latenz tritt ist ein wichtiger Parameter im Kontext des Video-Streamings für die Fernsteuerung von Maschinen. Sie beeinflusst sowohl die Benutzererfahrung als auch die Wirksamkeit der Aufgabenausführung. Obwohl die Benchmarks je nach Anwendung variieren, ist die Anpassung der Latenzanforderungen an die spezifische Art und die Anforderungen der jeweiligen Aufgabe wichtig.

3.2 Video-Streaming im Browser

In diesem Abschnitt liegt der Fokus auf dem Video-Streaming im Browser. Unterschiedliche Artikel, die sich mit Videostreaming-Lösungen im Browser befassen, werden zitiert, wobei besonders die Latenz hervorgehoben wird. Diese ist entscheidend, um bei der späteren Implementierung möglichst geringe Latenzen zu erreichen.

Yang et al. [YCK14] entwickelten eine Streaming-Lösung für Browser mit einer Latenz von mehr als fünf Sekunden. Für Apple-Geräte nutzten sie HLS, um einen Live-Stream im Browser zu ermöglichen. Im Jahr 2020 untersuchte Durak et al. [DAE+20] die Leistung von Apples HLS mit Unterstützung für niedrige Latenz, die 2019 eingeführt wurde. Sie stellten einen Vergleich mit der Low-Latency-DASH-Streaming-Lösung an. Ihre Ergebnisse zeigten eine Latenz von bis zu 300 Millisekunden. Dabei berücksichtigten sie auch, wie sich Netzwerkverkehr auswirkt. Ein möglicher Anwendungsfall von Apples HLS könnte darin liegen, den Stream einer großen Zuschaueranzahl bereitzustellen. Daher wurden CDN-Caching-Optionen in Betracht gezogen. Aufgrund der Konzeption von HLS könnte dies zu einer hohen Anzahl an Anfragen führen.

Eine weitere Untersuchung führten Essaili et al. [ELI18] durch, die ein Proof of Concept (PoC) für DASH (Dynamic Adaptive Streaming over HTTP) erstellten. Die niedrigste Latenz, die sie erreichen konnten, lag bei etwa 1000 Millisekunden. Sie beobachteten auch ein Jitter im Bereich von 100 bis

230 Millisekunden. Ein von ihnen identifiziertes Problem war die Verwendung von HTTP 1.1, das bei einzelnen GET-Anfragen hohe Latenzen aufweist. Sie schlugen vor, in zukünftigen Arbeiten ein Protokoll mit 0-RTT (null Rundlaufzeit) bei der Anfrageeinrichtung, wie QUIC, zu verwenden.

Koppehel und Pionteck [KP20] führten eine Implementierung von Videostreaming mit verschiedenen Technologien, nämlich HLS, HLS LL, MJPEG und ihren eigenen Lösungen SVS SW und SVS HW, durch. Ihre Video-Streams hatten eine Auflösung von 1920x1080 bei 90 fps. Mit HLS erreichten sie Latenzen von mehr als 1000 ms und mit MJPEG etwa 180ms. Ihre eigenen Lösungen, die eine verbesserte Leistung in den Bereichen Transport, Verkapselung und Dekodierung beinhalteten, zeigten in der reinen Software-Version eine Latenz von 100ms und in der FPGA-basierten Version eine Latenz von etwa 20ms.

Silhavy et al. [SPL+20] entwickelten eine Streaming-Lösung mit geringer Latenz für den Browser, basierend auf dash.js. Dies ist eine Open-Source-Bibliothek, die auf dem DASH-Konzept basiert. Da es, wie im Abschnitt 2.7 beschrieben, auf Chunks basiert, betrug die geringste in der Arbeit vorgestellte Latenz 2-3 Sekunden. Eine andere interessante Arbeit wurde von Shuai Zhao et al. [SZM16] durchgeführt. Sie adressierten das Problem der HTTP-Anfrage und RTT-1 mit der Idee, die Chunks in einem Push-basierten Modell mit Websockets zu senden. Obwohl sie nicht explizit eine Latenz für die Frames angeben, berichten sie von einer Übertragung von 170 ms für die Pakete.

Eine besonders umfassende Untersuchung wurde von Rodriguez-Gil et al. [ROGL18] durchgeführt. Sie erforschten verschiedene Streaming-Technologien, darunter MJPEG, MPEG-1 und MPEG-4, für einen Ultra-Low-Latency-Videostream. Sie analysierten verschiedene Metriken wie Frames pro Sekunde, Bandbreite, Browser-Unterstützung und die Leistung von Browser- und Server-Clients. Ihre Ergebnisse zeigten, dass alle Technologien in jedem Browser außer dem Internet Explorer funktionieren. Diese Studie wurde 2018 durchgeführt und heute, im Jahr 2023, ist der Internet Explorer veraltet. Um die Latenz zu messen, verwendeten sie eine IP-Webcam, die auf einen Desktop gerichtet war, auf dem eine Uhr zu sehen war. Ein Test-Laptop in der Nähe zeigte den Webcam-Feed basierend auf den Experimenteneinstellungen. Die Latenz wurde gemessen, indem die Live-Zeit auf der Desktop-Uhr und deren Darstellung auf dem Laptop verglichen wurde. MJPEG zeigte in ihren Tests die geringste Latenz von etwa 200ms, während MPEG-1 eine Latenz von 600ms und MPEG-4 eine von etwa 400ms hatte. Basierend auf ihren Ergebnissen kamen Rodriguez-Gil et al. [ROGL18] zu dem Schluss, dass MPEG-1 und MPEG-4 die besten Technologien für interaktive Anwendungen sind, insbesondere wenn hohe fps und geringe Bandbreite wichtig sind.

Abschließend untersuchte Lyko et al. [LBR+23] die Qualität des Erlebnisses (QoE) im Live-Streaming mit geringer Latenz. Ihr Hauptaugenmerk lag auf dem adaptiven Bitraten-Algorithmus (ABR), der schnell auf sich ändernde Netzwerkbedingungen reagieren kann. Sie beschrieben den ABR-Algorithmus Llama ausführlich, um dessen Leistung unter Bedingungen geringer Latenz zu zeigen. Auch wenn sie keine detaillierten Latenzdaten lieferten, erwähnten sie, dass sie unter bestimmten Bedingungen eine Latenz von nur 0,13s erreichten.

Diese Übersicht zeigt, dass es viele verschiedene Ansätze und Technologien gibt, um das Video-Streaming mit geringer Latenz im Browser zu ermöglichen. Die Wahl der besten Lösung hängt von den spezifischen Anforderungen und Rahmenbedingungen des jeweiligen Anwendungsfalls ab.

3.3 Qualitätsmessung von Video-Streaming

Hill et al. [HMH+09] präsentierte eine Methode zur End-to-End-Latenzmessung durch Erfassung eines Bildschirms, auf dem eine kontrollierte Szene abgespielt wird. Ihr Ziel war es, die End-to-End-Latenz für digitale und analoge Kameras zu messen und zu überprüfen, ob sie für den Einsatz bei Fernsteuerungen geeignet sind. Das Hauptkonzept bestand darin, einen Computerbildschirm zu erfassen, auf dem die Aufnahme selbst wiedergegeben wurde. Die Szene zeigte sowohl die „Szene“ als auch die Kamera-Videotransmission. Diese doppelte Funktion gewährleistete eine präzise Zeitmessung eventuell beobachteter Verzögerungen. Der Testaufbau begann mit einer roten Bildschirmanzeige, die zu einem grünen Quadrat überging. Das System analysierte die Videobilder, bis das Erscheinen des grünen Quadrats bestätigt wurde. Die Latenz, also die Zeit von der Anzeige bis zur Erkennung, wurde bei jedem Testlauf protokolliert. Diese Methodik erwies sich als vielseitig und war anwendbar auf verschiedene Kamera-Spezifikationen, Auflösungen, Komprimierungseinstellungen und Netzwerkconfigurationn. Leider wurde nicht erwähnt, wie genau sie den Unterschied berechneten oder wie sie die Unterschiede erkannten und die Latenzmessungen vornahmen.

Jansen und Bulterman [JB13] stellte einen Ansatz zur Messung der End-to-End-Video-Latency vor, bei dem die zugrundeliegende Videopipeline bzw. die Videoübertragung von der eigentlichen Messung unabhängig ist. Ihr Aufbau besteht aus folgenden Teilen:

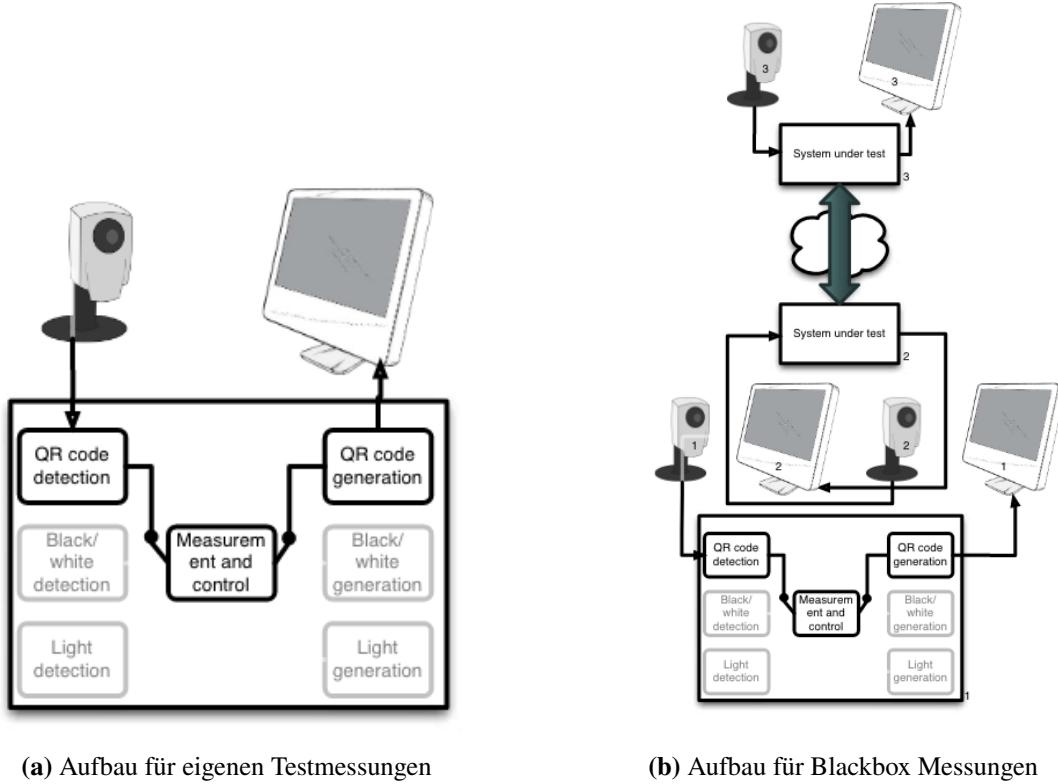
Zunächst gibt es einen Selbstmessungsaufbau, der in Abbildung 3.1a dargestellt ist und hauptsächlich zur Kalibrierung dient. Die Kamera des Messsystems ist auf den eigenen Bildschirm gerichtet und erfasst die inhärenten Verzögerungen des Messsystems selbst. Dies schließt Verzögerungen ein, die durch die Hardwarekomponenten, das Betriebssystem und andere Faktoren eingeführt werden. Die Kalibrierungsmessung liefert den Durchschnitt und die Standardabweichung der Verzögerungen im Messsystem, die nicht automatisch kompensiert werden können [JB13].

Danach gibt es den eigentlichen Aufbau als Blackbox-Messsystem, dargestellt in Abbildung 3.1b. Ein QR-Code wird auf einem Bildschirm (Bildschirm 1) angezeigt und von einer Kamera (Kamera 2) erfasst. Der QR-Code durchläuft das zu testende System und erscheint dann auf einem zweiten Bildschirm (Bildschirm 2). Dieser auf Bildschirm 2 angezeigte QR-Code wird von einer anderen Kamera (Kamera 1) zur Messung erfasst [JB13].

Für die eigentliche Messung wird ein drittes System verwendet, welches das "Blackbox-Setup" verarbeitet, auch als videoLat-Tool bezeichnet. Das videoLat-Tool verarbeitet die Daten, um die Round-Trip-Verzögerung des zu testenden Systems zu bestimmen. Das Tool ist so konzipiert, dass es automatisch die inhärenten Verzögerungen des Messsystems berücksichtigt, um sicherzustellen, dass die Ergebnisse die tatsächliche Latenz des getesteten Systems widerspiegeln.

Die Einweg-Verzögerung wird gemessen, indem das sendende videoLat-System auf die Generierung von QR-Codes mit einer festgelegten Frequenz im freien Lauf eingestellt wird und das empfangende videoLat-System in den reinen Empfangsmodus versetzt wird. Nach der Messung wird eine Offline-Verarbeitung der gespeicherten Zeitstempel beider videoLat-Systeme verwendet, um die Einweg-Verzögerung zu berechnen.

Kryczka et al. [KAN13] präsentieren ein Tool namens "AvCloak", das darauf abzielt, wichtige Leistungsmetriken in proprietären VC-Anwendungen zu messen, wie Mund-zu-Ohr-Latency (MEL), Erfassungs-bis-Anzeige-Latency (CDL) und die Audio-Video-Synchronisationsabweichung. AvCloak


Abbildung 3.1: Aufbauten von videoLat Messungstool

simuliert eine WebCam, indem es einen Video-Stream mit kodierten Zeitstempeln erzeugt und diese Daten über die verschiedenen Videostream Software an die Empfängerseite überträgt. Auf der Empfängerseite dekodiert AvCloak die Zeitstempel aus den Medienausgaben. Der Aufbau wird im Forschungspapier detailliert beschrieben. Für die Initiierung auf der Senderseite wird die Uhr des Senders mit der des Empfängers synchronisiert, die Konferenzeinladung versendet, ein Audioeingabe-Synthesethread gestartet, PulseAudio für Audio-Loopback konfiguriert, ein Videoeingabe-Synthesethread eingeleitet und eine GStreamer-Pipeline für Video-Loopback eingerichtet. Auf der Empfängerseite läuft der NTP-Server-Daemon¹ ntpd im Hintergrund, die Konferenzeinladung wird akzeptiert, ein Audioausgabe-Analysethread gestartet, PulseAudio für Audio-Loopback eingerichtet und ein Videoausgabe-Analysethread gestartet.

Die eigentliche Videoverzögerungsmessung erfolgt durch Synchronisierung einer Uhr und das Senden des EncodeZeitstempels in einem Barcode. Auf der Empfängerseite wird dieser Barcode dann vom synchronisierten Zeitstempel subtrahiert und die Differenz berechnet.

Jennehag et al. [JFF16] präsentierte eine Lösung für Video-Streaming mit geringer Latenz. Dafür wählten die Autoren einen unkomplizierten Ansatz für die Messung der Latenz. Dadurch hatten sie den Vorteil, aufgrund ihrer Anordnung die zugrunde liegende Videopipeline oder Netzwerkein-

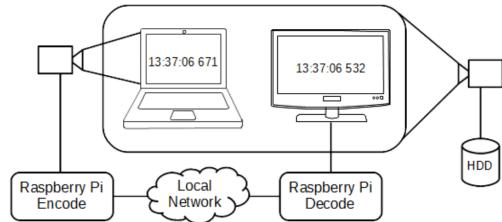
¹ntp steht für *Network Time Protocol*, ein Protokoll, das verwendet wird, um die Uhren von Computern über ein Netzwerk zu synchronisieren.

3 Literaturrecherche

richtung zu ändern. Sie verglichen das Video manuell mit der Aufnahme und der Anzeige durch Aufzeichnung eines Zeitstempels. Dazu erfassten die Autoren das Video und nahmen es dann mit einer zweiten Kamera mit 300 fps auf. Die Autoren wählten diese Methode um die vollständige Verzögerung von der Aufnahme bis zur Anzeige zu messen. Die Autoren gaben nicht an, wie sie ihr Video analysierten bzw. wie sie die Latenz aus den Barcodes berechnet haben. Zum Beispiel ob sie jeden Zeitstempel manuell notierten und dann den Unterschied berechneten oder ein automatisches Tool verwendeten. Obwohl sie eine Kamera mit 300 Bildern pro Sekunde (FPS) zur Bewertung ihrer Einrichtung verwendeten, zeigt der von ihnen erwähnte Monitor nur 60 Hz an. Daher konnten sie nur Änderungen jede $\frac{1}{60}$ Sekunde erkennen, nicht jede $\frac{1}{300}$, wie es die Kamera ermöglichen würde. Das bedeutet nämlich auch, dass sie maximal $\frac{1}{60} = 16,67$ ms erkennen konnten. Als Ergebnis für einen Video Stream innerhalb eines Netzwerkes konnten sie eine Latenz von 163ms erreichen.



(a) Bild vom Latenzmessungs Aufbau



(b) Skizze für Latenzmessungs Aufbau

Abbildung 3.2: Aufbauten von Low Delay Video Streaming

Ubik und Pospisilik [UP21] verfolgen ursprünglich das Ziel die Latenz von Videokameras zu messen. Sie verfolgen nicht das Ziel die Videolatenz einer Videostreaming-Lösung zu erfassen. Die Autoren wenden drei verschiedene Methoden an, um die Latenz einer Videokamera zu messen. Zwei dieser Methoden könnten jedoch auch eine Videoübertragung einschließen und somit als mögliche Verfahren zur Messung der Latenz in einem Videostreaming-Setup dienen. Eines der Setups umfasst einen Photodetektor, der an ein Oszilloskop angeschlossen ist, eine Videokamera mit einem analogen Konverter, ebenfalls verbunden mit einem Oszilloskop, sowie das Oszilloskop selbst und eine Taschenlampe. Wenn die Lichtquelle blinkt, erzeugen sowohl der Photodetektor als auch der analoge Videoeingang Wellenformen mit Hoch- und Tiefpunkten auf dem Oszilloskop, die zur Berechnung der Latenz verwendet werden können.

Diese illustriert eine angepasste Konfiguration, die potenziell für die Implementierung der Videostreaming-Technologie (markiert als grüne Box) geeignet ist. Der vorgestellte Versuchsaufbau umfasst eine Taschenlampe, einen Photodetektor, eine Kamera und einen Monitor. Hierbei wird der Photodetektor direkt neben dem Monitor platziert, während die Kamera auf die Taschenlampe ausgerichtet ist. Ein Zähler löst die Taschenlampe aus, was einen sichtbaren Blitz auf dem Monitor erzeugt. Für die Kontrolle einer blinkenden LED, das Auslesen des Photodetektors und die Berechnung der Latenz mit einem Zähler kann ein Arduino eingesetzt werden [UP21].

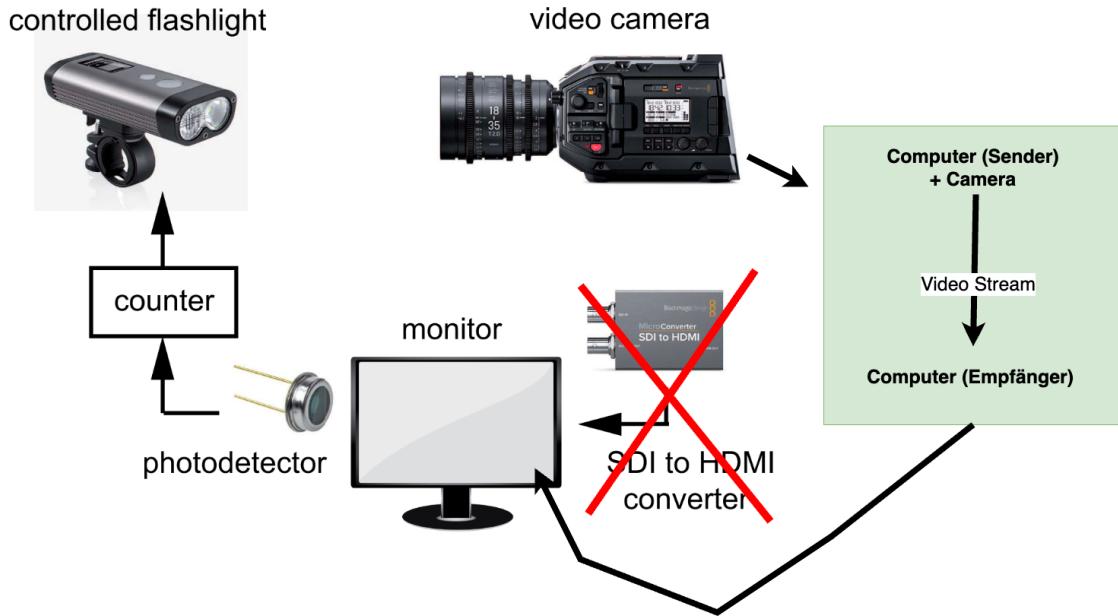


Abbildung 3.3: Modifizierter Aufbau

Des Weiteren wurde eine Methode vorgestellt, die als Wellenformverschiebung bezeichnet wird. Der wesentliche Unterschied dieser Methode liegt darin, dass der Photodetektor an ein Oszilloskop angeschlossen ist. Dort wird die Spannungsstärke gemessen, die sich bei Lichteinfall verändert. Diese Veränderung ermöglicht es, die Latenzzeit zu ermitteln - also die Zeitspanne vom Moment der Stromzufuhr zum Blitzlicht bis zum Messpunkt am Photodetektor.

3.4 Augmentation - Tracking & Mapping

In dieser Arbeit wird ein Prototyp entwickelt, dessen Ziel es ist, ein CAD 3D-Modell auf ein reales Modell zu mappen. Dieser Ansatz ist besonders relevant in der industriellen Anwendung von Augmented Reality (AR), wo die exakte Überlagerung von virtuellen Konstruktionsdaten über physische Werkstücke von entscheidender Bedeutung ist. Die Technologie soll nicht nur die Visualisierung von geplanten Schweißnähten vor dem Laserschweißprozess ermöglichen, sondern auch die Integration der Augmentation in den finalen Video-Stream leisten. Hierbei konzentriert sich der Prototyp darauf, im ersten Schritt das 3D-Modell sichtbar zu machen und in weiteren Schritten zusätzliche Informationen bereitzustellen. Für eine erfolgreiche Implementierung werden Methoden erforscht, die eine präzise Übereinstimmung der virtuellen CAD-Modelle mit der Position der realen Objekte gewährleisten.

Wuest und Stricker [WS07] beschreiben in Ihrer Arbeit den "modellbasierten" Ansatz, der im Wesentlichen die folgenden Schritte umfasst: die Erstellung eines Linienmodells aus dem CAD-Modell, gefolgt von der Ausrichtung dieses Linienmodells mit einem 2D-Bild. Die Autoren nutzen Bildgradienten, um die Ausrichtung zu maximieren und so die optimale Position zu bestimmen. Anschließend wird die Kameraposition unter Verwendung von Daten aus dem Bild und dem

3 Literaturrecherche

Linienmodell ermittelt. Aufbauend darauf erweitert Wuest et al. [WEW+16] die frühere Arbeit des Autors aus [WS07]. In diesem Artikel wendet der Autor nicht nur die Ausrichtungsmethode seiner vorherigen Arbeit an, sondern erweitert deren Anwendungsbereich, indem er die direkte Verwendung von CAD-Modellen ermöglicht. Zusätzlich demonstriert er eine Anwendung, die 3D-CAD-Modelle in Echtzeit auf reale Objekte abbildet.

In Song et al. [SCH+20] stellen die Autoren eine Methode vor, mit der es einem Roboter ermöglicht werden soll, sich erfolgreich mit einer Betankungsvorrichtung zu verbinden. Der zugrunde liegende Ansatz ist die "Moving-EdgeMethode, die auf dem Erkennen der Konturen des realen Objekts basiert. Diese Technik beinhaltet das Projizieren des CAD-Modells des Objekts auf die Bildebene und das Verfolgen der sichtbaren Kanten, wodurch eine genaue Ausrichtung mit dem realen Objekt ermöglicht wird. Das System verwendet virtuelles visuelles Servoing, um die Bewegungen des Roboters dynamisch anzupassen und eine präzise Andockung zu gewährleisten, indem die Diskrepanz zwischen dem projizierten CAD-Modell und den tatsächlichen Bildmerkmalen minimiert wird.

4 Web-based Streaming

In diesem Kapitel werden potenzielle Lösungen für die Implementierung von Echtzeit-Videostreaming, das in einem Webbrower dargestellt werden kann, vorgestellt. Zu Beginn erläutern wir die notwendigen Anforderungen, die eine geeignete Streaming-Lösung erfüllen sollte. Anschließend erfolgt eine detaillierte Betrachtung und Analyse der verfügbaren Technologien sowie Software-Tools. Abschließend vergleichen wir die verschiedenen Lösungsalternativen anhand zuvor definierter Kriterien, um eine fundierte Auswahl treffen zu können.

4.1 Anforderungen für das webbasierte Streaming

Diese Anforderungen, die teils spezifisch für Trumpf sind und teils allgemeine Relevanz besitzen, stammen aus den aktuellen Gegebenheiten, die beim aktuellen Produkt von TRUMPF Laser vorherrschen, sowie aus Gesprächen mit dem Team bei TRUMPF Laser. Ein wesentlicher Aspekt dieses Produkts ist die Betriebsüberwachung von industriellen Lasermaschinen. Dies bedeutet, dass der Laserprozess kontinuierlich überwacht werden muss, um eine optimale Leistung und Sicherheit zu gewährleisten. Diese Überwachungsaufgaben sind entscheidend für die Funktionsfähigkeit und Zuverlässigkeit der Maschinen. Die abgeleiteten Anforderungen dienen im weiteren Verlauf der Arbeit als Grundlage für die Beurteilung der Streaming-Möglichkeiten.

Echtzeit Video Streaming

Für die Echtzeit-Überwachung ist eine Übertragung mit minimaler Latenz erforderlich, idealerweise ohne Buffering. Minimale Latenz ist für die Fernsteuerung der Maschinen essentiell, um den Bedienern ein sofortiges Eingreifen in die Maschinenfunktionen zu ermöglichen.

Implementierungsaufwand

Der technische Aufwand für die Implementierung ist ein weiteres wichtiges Kriterium. Es beurteilt die Komplexität und die benötigten Ressourcen für die Integration der Streaming-Technologie in die existierende Software. Diese Beurteilung gibt Aufschluss über die Machbarkeit und Skalierbarkeit der vorgeschlagenen Lösung.

Unterstützung mehrerer Clients (TRUMPF Spezifisch)

Das Systemdesign muss die simultane Verbindung mehrerer Nutzer unterstützen. Mehrere Clients sollen gleichzeitig auf den Livestream zugreifen können, ohne Einbußen bei Leistung oder Qualität – eine Anforderung, die sowohl aus Nutzerperspektive als auch für Überwachungszwecke kritisch ist.

Systemzuverlässigkeit

In einem industriellen Umfeld, wo Betriebsstillstand zu erheblichen Verzögerungen und finanziellen Verlusten führen kann, ist Zuverlässigkeit von größter Bedeutung. Das Streaming-System muss dauerhaft stabil laufen und hohe Zuverlässigkeit gewährleisten.

Browserbasiertes Streaming

Die Streaming-Lösung muss kompatibel mit Webbrowsern sein. Dies impliziert, dass der Benutzer in der Lage sein muss, den Videostream direkt im Webbrowser abzuspielen. Eine solche Kompatibilität erlaubt es den Nutzern, auf den Stream zuzugreifen, ohne spezielle Software installieren zu müssen. Diese Flexibilität, die ein Webbrowser bietet, ist ein entscheidender Vorteil.

Verlustfreie Bildqualität (TRUMPF Spezifisch)

Es ist von entscheidender Bedeutung, dass während der Übertragung eine hohe Bildqualität gewährleistet wird, um den Verlust wichtiger visueller Informationen zu vermeiden. Zudem kann, abhängig von der Art der Implementierung, der Einsatz einer potenziell verlustfreien Codierung für spätere Qualitätssicherungsmaßnahmen wichtig sein.

Ressourcenauslastung (TRUMPF Spezifisch)

Die auf Kundenseite eingesetzten Computer verfügen möglicherweise über begrenzte Rechenkapazitäten, daher ist eine geringe Rechenlast erforderlich. Eine optimierte Performance, die gleichzeitig die Systemanforderungen an Kundencomputer reduziert.

Netzwerkbandbreite

Eine optionale Überlegung für das Systemdesign ist die Berücksichtigung variierender Netzwerkbandbreiten, vor allem bei WLAN-Verbindungen, die zu Bandbreitenschwankungen neigen. Der Einsatz von Technologien wie der adaptiven Bitratenübertragung könnte dabei helfen, unter unterschiedlichen Netzwerkbedingungen eine stetig stabile und zuverlässige Verbindung zu gewährleisten.

4.2 Beschreibung und Implementierung der Streaming-Lösungen

In diesem Abschnitt wird eine detaillierte Übersicht über verschiedene Technologien präsentiert, die das Video-Streaming in Webbrowsern ermöglichen. Jede Streaming-Technologie wird zunächst umfassend beschrieben und anschließend deren Implementierungsmöglichkeiten erläutert.

Beginnend mit der Darstellung einer Methode, die auf Einzelbildern (Single Images) basiert, liegt ein besonderer Fokus auf MJPEG¹ zur Realisierung eines Video-Streams im Browser. Danach wird Web Real-Time Communication (WebRTC)² vorgestellt, eine Technologie, die nativ in modernen Webbrowsern implementiert ist. Weiterhin wird erläutert, wie MPEG-Technologien im Browser genutzt werden können, einschließlich der Technologien HLS und MPEG-DASH. Ein charakteristisches Merkmal dieser Technologien ist ihre Fähigkeit, sich dynamisch an die Bandbreite und Leistung des jeweiligen Clients anzupassen.

Tabelle 4.1 bietet einen strukturierten Überblick über die besprochenen Technologien. Sie stellt die spezifischen Eigenschaften und Unterscheidungsmerkmale jeder Technologie dar, um eine vergleichende Betrachtung zu erleichtern. Die in der Tabelle aufgeführten Attribute dienen als Grundlage für die nachfolgenden detaillierten Beschreibungen und Implementierungsbeispiele.

Es ist wichtig zu verstehen, dass jede Streaming-Methode spezifische Vorzüge und Einschränkungen hat. Dieses Verständnis ist entscheidend für die Auswahl der passenden Technologie für geplante Streaming-Projekte. Die folgenden Abschnitte widmen sich daher jeder dieser Technologien im Detail und bieten Einblicke in ihre erfolgreiche Implementierung.

4.2.1 Einzelbilder & MJPEG

In dieser Methode wird ein Video durch eine schnelle Abfolge einzelner Bilder erzeugt. Der Handlungsspielraum entsteht dabei sowohl durch die Auswahl des Formats als auch durch die des Übertragungswegs. Verschiedene Formate, die sich in ihrer Kompressionsart – verlustfrei oder verlustbehaftet – unterscheiden, stehen zur Verfügung. Verlustfreie Formate, wie PNG, erhalten die Bildqualität, während Formate wie JPEG Kompression nutzen, um Bandbreite zu sparen. In einigen Fällen lässt sich die Kodierung der Bilddaten durch den Einsatz von Hardware-Beschleunigern optimieren, was besonders bei verbreiteten und gut unterstützten Formaten wie Motion JPEG vorteilhaft ist.

Für den Datentransport kommen unterschiedliche Methoden in Betracht:

- WebSockets³ bieten einen kontinuierlichen Kanal für interaktive Kommunikation zwischen Browser und Server.
- Der Content-Type multipart mit einem Boundary ermöglicht es, mehrere Nachrichtenteile in einer einzigen HTTP-Anfrage zu übermitteln [ROGL18].

¹todo: <https://datatracker.ietf.org/doc/html/rfc2435>

²WebRTC ermöglicht Browser bzw. deren Websites die Implementierung einer Live Direkt Verbindung von Video und Audiomedien. [Moz23h]

³Die WebSocket API ermöglicht das Aufbauen einer bidirektionalen Datenverbindung im Browser [Moz23e].

4 Web-based Streaming

Listing 4.1 Beispiel Skript für Senden Einzelbilder über Http-Stream

```
import ...

camera = cv2.VideoCapture(0)
# Setzen der Bildbreite
camera.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
# Setzen der Bildhöhe
camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
# Setzen der gewünschten Bildrate (Frames per Second)
camera.set(cv2.CAP_PROP_FPS, 30)

def encode_frame():
    while True:
        # Bild von der Kamera lesen
        success, frame = camera.read()
        if not success:
            break
        else:
            # Das Bild in JPEG-Format kodieren
            ret, buffer = cv2.imencode(".jpg", frame)
            # Umwandlung des kodierten Bildes in Bytes
            frame = buffer.tobytes()

            # Endzeit für die Zeitmessung
            end_time = time.time()

            # Generieren des Antwort-Streams im multipart/x-mixed-replace Format
            yield (b"--frame\r\n" b"Content-Type: image/jpeg\r\n\r\n" + frame + b"\r\n")

@app.route("/video_feed")
def video_feed():
    # Streamen des Videos Frame für Frame und Trennung durch bonoundary
    return Response(
        encode_frame(), mimetype="multipart/x-mixed-replace; boundary=frame"
    )
```

- Server-Sent-Events⁴ sind eine Browser API, um automatische Updates vom Server zum Browser in einem stetigen Strom zu senden.

Die Abbildung 4.1 zeigt eine Streaming-Architektur, auf Basis von einer Einzelbildübertragung.

Hinsichtlich der Bildformate unterstützen Browser verschiedene Typen, wie in der Dokumentation von mdn⁵ aufgeführt [Moz23c]. Neben JPEG, das für seine Effizienz bekannt ist, existiert PNG, das eine hohe Bildqualität ohne Verluste gewährleistet, sowie moderne Formate wie WebP, deren Verfügbarkeit je nach Browser variieren kann.

⁴Eine Methode, bei der der Server kontinuierlich Daten an den Client sendet, innerhalb einer persistenten, eindirektonalen Verbindung.

⁵MDN (Mozilla Developer Network) ist eine umfassende Online-Ressource für Entwickler, die Informationen, Dokumentation und Ressourcen rund um Webtechnologien wie HTML, CSS und JavaScript bietet.

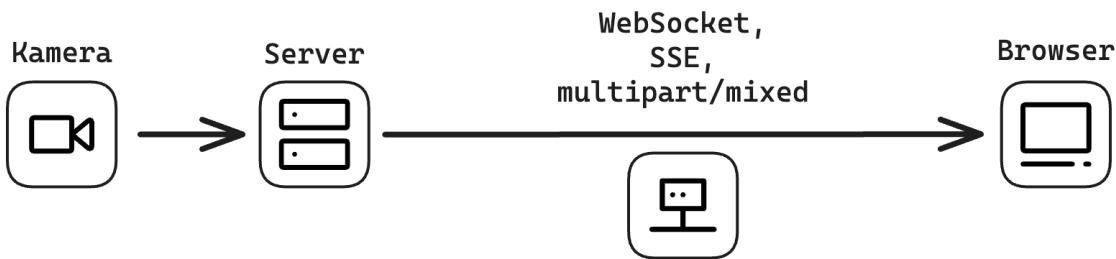


Abbildung 4.1: Darstellung einer Einzelbild-Video-Streaming-Architektur

Implementierung

Das Codebeispiel 4.1 demonstriert eine einfache Implementierungsform, bei der Einzelbilder gesendet werden. Je nach Anforderung lässt sich das Format variieren, um beispielsweise Bilder im .bmp-Format zu übertragen. Wichtig zu erwähnen ist, dass die eigentliche Datenübertragung über einen standardmäßigen HTTP-Request erfolgt, insbesondere unter Verwendung von "multipart/x-mixed-replace; boundary=frame", wobei jedes Bild durch den "boundary" d.h. der Separator getrennt wird.

Eine weitere Methode, die sich im Datentransport unterscheidet, ist WebSocket. Abbildung 4.2 zeigt diese Methode.

Zusammenfassend ermöglicht die Implementierung der Bildübertragung mittels einzelner Bilder eine unkomplizierte Lösung, die eine hohe Kompatibilität sowohl bei den Kodierungsformaten als auch bei den Übertragungstechniken bietet. Jedoch sollte man bei dieser Methode die hohe Bandbreitennutzung aufgrund der fehlenden Interframe-Kompression sowie das Fehlen von Quality of Service (QoS) Mechanismen bedenken, welche eine konstante Übertragungsqualität sicherstellen könnten.

4.2.2 WebRTC

WebRTC ist eine Peer-to-Peer-Kommunikationstechnologie, die direkt in moderne Webbrowser integriert ist [Can23]. Diese Technologie hat sich zu einem wesentlichen Bestandteil vieler Kommunikationsanwendungen für Audio-, Video- und Datentransfer entwickelt. Besonders für Videotelefonie ist WebRTC sehr gut geeignet. Als Open-Source-Projekt basiert WebRTC auf verschiedenen JavaScript-APIs, die es ermöglichen, Kommunikation ohne zusätzliche Plugins oder externe Anwendungen zu starten und aufrechtzuerhalten. Die Kernarchitektur von WebRTC besteht aus verschiedenen Elementen, darunter Signaling- und TURN-Server.

Abbildung 4.2 zeigt eine vereinfachte Darstellung des typischen Aufbaus zwischen zwei Peers.

Die Funktion eines Signaling-Servers: Der erste Kontakt zwischen zwei Geräten wird durch einen Prozess namens *Signaling* erleichtert. Ein Signaling-Server nutzt die WebRTC-Signaling-Mechanismen, um Medieninformationen auszutauschen und die Kommunikationsitzung zu koordinieren. Dieser handhabt den Austausch von Angebot- und Antwort-Nachrichten, welche die vorgeschlagenen

4 Web-based Streaming

Listing 4.2 Beispielskript zum Senden von Einzelbildern über WebSockets

```
import ...

# Kamera ini
camera = cv2.VideoCapture(0)
camera.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
camera.set(cv2.CAP_PROP_FPS, 10)

async def video_stream(websocket, path):
    while True:
        start_time = time.time() # Startzeit fÃ¼r die Zeitmessung
        success, frame = camera.read() # Bild von der Kamera lesen
        if not success:
            print("Failed to grab frame")
            break
        ret, buffer = cv2.imencode(".jpg", frame) # Das Bild in JPEG-Format kodieren
        frame = buffer.tobytes() # Umwandlung des kodierten Bildes in Bytes
        end_time = time.time() # Endzeit fÃ¼r die Zeitmessung

        # Send the frame over the WebSocket
        try:
            await websocket.send(frame)
        except websockets.exceptions.ConnectionClosed:
            print("Connection closed")
            break

    # Start the WebSocket server
    start_server = websockets.serve(video_stream, "", 3000)

asyncio.get_event_loop().run_until_complete(start_server)
asyncio.get_event_loop().run_forever()
```

Medienformate und Kommunikationspfade beschreiben. Der Signaling-Server übermittelt auch Sitzungssteuerungsnachrichten, Netzwerkinformationen und Fehlermeldungen zwischen den kommunizierenden Peers [SSP15].

Interaktionen mit TURN-Servern: In Situationen, in denen Peer-to-Peer-Kommunikation durch NAT (Network Address Translation) oder Firewalls eingeschränkt ist, fungiert ein TURN-Server (Traversal Using Relays around NAT) als Mittelsmann, um Mediendaten zwischen den beiden Parteien zu übertragen [SSP15].

Unterstützung von Video-Codecs in WebRTC: Die Qualität und Effizienz des Video-Streamings hängt stark von den verwendeten Codecs ab. WebRTC unterstützt verschiedene Video-Codecs, einschließlich VP8 und VP9 für hochkomprimiertes Video-Streaming, sowie H.264, welches wegen seiner Balance zwischen Kompression und Qualität beliebt ist. Die Codec-Auswahl beeinflusst Faktoren wie Latenz, Bandbreitennutzung und die Gesamtbildqualität.

4.2 Beschreibung und Implementierung der Streaming-Lösungen

Sicherstellung der Dienstqualität (QoS): Die Dienstqualität ist für Echtzeitkommunikation entscheidend. WebRTC implementiert QoS durch die Optimierung der Datenübertragung abhängig von den verfügbaren Netzwerkbedingungen. Es kann die Bitrate von Video- und Audiostreams anpassen und in beschränkten Umgebungen Flüssigkeit gegenüber Qualität bevorzugen. Weiterhin nutzt WebRTC das Real-Time Transport Control Protocol (RTCP) Feedback, um die Leistung der Verbindung zu verwalten und zu überwachen [BSTD17].

```
// Zugriff auf Media Device
navigator.mediaDevices.getUserMedia({ video: true, audio: true })
  .then(stream => {
    // Stream im lokalen Video-Element anzeigen
    const localVideo = document.getElementById('localVideo');
    if (localVideo) {
      localVideo.srcObject = stream;
    }

    // Peer-Connection erstellen
    const peerConnection = new RTCPeerConnection();

    // Stream zur Peer-Connection hinzufügen
    stream.getTracks().forEach(track => peerConnection.addTrack(track, stream));

    // Fernvideo-Stream behandeln
    peerConnection.ontrack = event => {
      const remoteVideo = document.getElementById('remoteVideo');
      if (remoteVideo) {
        remoteVideo.srcObject = event.streams[0];
      }
    };

    // Signaling-Mechanismus (muss implementiert werden)
    // Beispiel: socket.emit('offer', offer);
    peerConnection.createOffer()
      .then(offer => peerConnection.setLocalDescription(offer))
      .then(() => {
        // Angebot an den entfernten Peer senden
        // (Hier muss ein Signaling-Server eingesetzt werden)
      });

    // Antwort des entfernten Peers behandeln
    // Beispiel: socket.on('answer', answer => {...});
  })
  .catch(e => console.log(`Error: ${e.message}`));
```

Listing 4.1: Beispiel für die Nutzung der WebRTC-API in einer JavaScript-Anwendung

Code Beispiel 4.1 wird gezeigt, wie eine einfache WebRTC-basierte Videoübertragung in einer JavaScript implementiert werden kann. Es ist wichtig zu beachten, dass für das vollständige Funktionieren dieser Anwendung zusätzlich ein Signaling-Server erforderlich ist. In diesem Beispiel wird dies z.B. durch "socket.on(...)" gezeigt. Ein solcher Server kann optimalerweise

4 Web-based Streaming

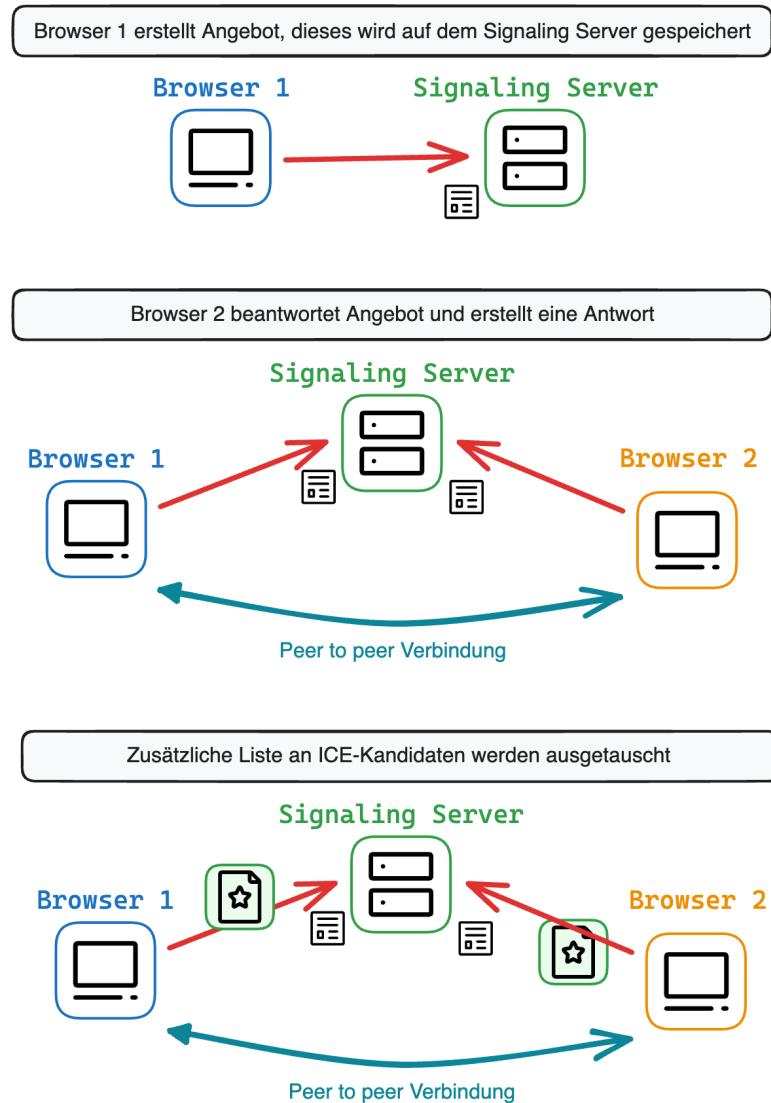


Abbildung 4.2: Vereinfachter Beispielhafter Aufbau einer WebRTC Verbindung und deren Komponenten

als WebSocket-Server realisiert werden. Der Signaling-Server spielt eine entscheidende Rolle im Verbindungsauflauf zwischen den Peers, indem er Angebot- und Antwort-Nachrichten sowie andere Signaling-Informationen übermittelt.

In der Analyse des Streaming wird das Tool *Camera Streamer*⁶ verwendet. Dieses Tool ermöglicht das Streamen von über WebRTC von einem beligen Linux Gerät.

⁶Siehe <https://github.com/ayufan/camera-streamer>

4.2.3 JSMPEG (Demuxer)

JSMPEG⁷ ist ein Open-Source-Videoplayer, der in JavaScript entwickelt wurde. Seine Hauptkomponenten umfassen einen MPEG-TS-Demultiplexer sowie Decoder für MPEG-1-Video und MP2-Audio. Ein Demultiplexer hat die Funktion, einen eingehenden Stream, der mehrere Signale oder Datenströme kombiniert, in individuelle, separate Datenströme aufzuteilen. Im Falle einer MPEG-TS-Datei werden beispielsweise Audio- und Videodaten in einen einzigen Stream gemischt; der Demultiplexer von JSMPEG separiert diese in eigenständige Audio- und Videostreams, sodass diese korrekt verarbeitet werden können [Dom23]. Ein Decoder konvertiert die komprimierten Videodaten – hier MPEG1 – in eine Serie von Bildern, die anschließend angezeigt werden können. Für die Darstellung zieht JSMPEG WebGL heran, während die Audiowiedergabe über die WebAudio-API erfolgt. Darüber hinaus unterstützt JSMPEG das Streaming, wobei Puffergrößen für Audio- und Videodaten individuell angepasst werden können. Zudem bietet es Callbacks für die Ereignisse der Video- und Audiodekodierung. Der Datentransfer erfolgt via WebSocket, was für eine effiziente Übertragung sorgt [Dom23]. Eine detaillierte Anleitung und Dokumentation zur Verwendung und Konfiguration von JSMpeg ist auf der GitHub-Seite des Projekts unter <https://github.com/phoboslab/jsmpeg> verfügbar.

4.2.4 HTTP Live Streaming

HLS, ist eine von Apple Inc. entwickelte Technologie zur Übertragung von Audio- und Videodaten über Webserver an iOS-basierte Endgeräte. Diese Technologie nutzt HTTP, ein verbreitetes Protokoll des Internets, als Grundlage für den Datenaustausch. HLS zeichnet sich durch eine Reihe fortgeschrittener Funktionen aus. Dazu gehören die Ausstrahlung von Live-Sendungen sowie das Bereitstellen vorproduzierten Inhalts, bekannt als Video-on-Demand (VOD). Weiterhin ermöglicht HLS mehrere alternative Übertragungsströme mit unterschiedlichen Bitraten und passt diese intelligent an schwankende Netzwerkbandbreiten an. Sicherheitsmaßnahmen wie Medienverschlüsselung und Nutzerauthentifizierung sind ebenfalls integrierte Bestandteile von HLS. Grafik 4.3 veranschaulicht den Prozess, wie Audio- und Videomaterial zum Client gestreamt werden kann. Zunächst wird der Eingangsdatenstrom vom Server in ein kompatibles Format codiert, das sich für Streaming-Anwendungen eignet. Darüber hinaus wird der Inhalt in kleine, für die Verteilung optimierte Segmente aufgeteilt, was allerdings zu einer Erhöhung der Latenz führen kann, da die einzelnen Segmente sequenziell geladen werden müssen. Nach der Segmentierung werden die Dateien über HTTP bereitgestellt, vom Client abgerufen und schließlich wiedergegeben.

4.2.5 MPEG-DASH

MPEG-DASH, auch bekannt als Dynamic Adaptive Streaming over HTTP (DASH), ist eine Technologie, die den Transfer von Audio- und Videoinhalten (Audio und Video (AV)) über das Hypertext Transfer Protocol (HTTP) ermöglicht. Das grundlegende Konzept von MPEG-DASH ähnelt dem von HTTP Live Streaming (HLS): Ein Server teilt einen Audio-Video-Stream in Segmente auf, die dann über einen Webserver den Clients zur Verfügung gestellt werden [Sto11]. Ein zentrales Merkmal von MPEG-DASH ist das adaptive Streaming. Dabei kodiert der Server die

⁷Ein MPEG1 Videoplayer geschrieben in Javascript, verfügbar auf <https://github.com/phoboslab/jsmpeg>

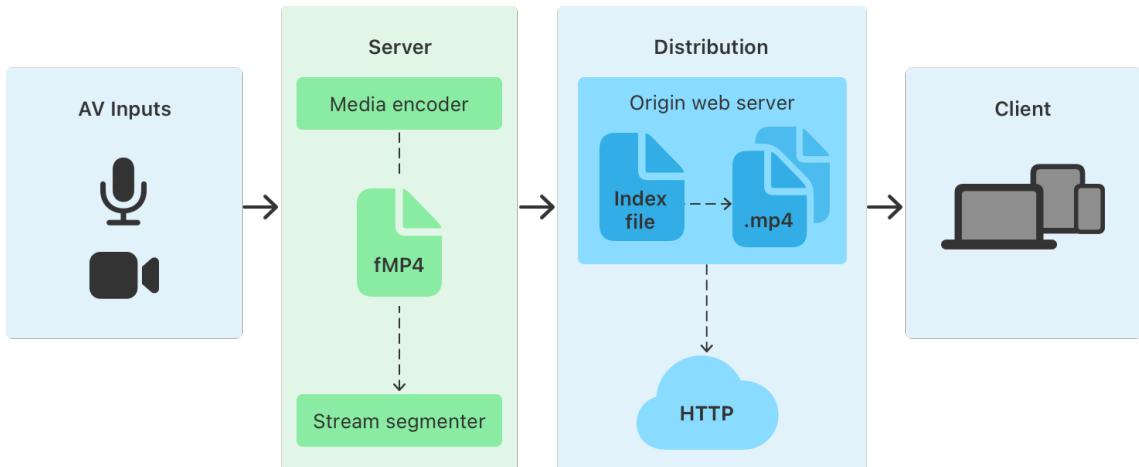


Abbildung 4.3: Aufbau von HLS [App23]

Segmente in verschiedenen Qualitätsstufen und der Videoplayer auf der Client-Seite kann zwischen diesen Segmenten nahtlos und ohne Unterbrechung des Videos wechseln, abhängig von der aktuellen Netzwerkanforderungen und der Leistung des Wiedergabegeräts. Im Vergleich zu HLS bietet MPEG-DASH die Flexibilität, verschiedene Codierungsformate für die Segmente zu nutzen. Die Segmente selbst sind üblicherweise zwischen zwei bis zehn Sekunden lang; Forschungen wie die von Shuai Zhao et al. [SZM16] experimentieren jedoch mit noch kürzeren Segmenten, um die Verzögerungszeit weiter zu reduzieren. Hierbei wird auch der Austausch des Transferprotokolls erforscht, um die Round-Trip-Time (RTT) zu minimieren, da das Abrufen mittels HTTP-Get-Methoden immer eine Anfrage und eine Antwort nach sich zieht. Das Media Presentation Description (MPD)-File, ein weiteres wichtiges Element von MPEG-DASH, enthält Informationen über die Mediensegmente und deren Verfügbarkeit. Diese XML-Datei beschreibt die Struktur des Content Streams, inklusive der Metadaten für die einzelnen Segmente und die verfügbaren Qualitätsstufen, wodurch das adaptive Streaming ermöglicht wird. Der MPD-Datei kommt somit eine Schlüsselrolle im DASH-Streaming-Prozess zu, da sie dem Client hilft, die richtigen Segmente zur richtigen Zeit herunterzuladen und dadurch eine optimierte Wiedergabequalität gewährleistet [SZM16].

4.3 Entwicklung eines Streaming-Prototypen

Im Rahmen dieser Masterarbeit wurde ein Prototyp für Streaming-Anwendungen entwickelt. Abbildung 4.4 veranschaulicht den Aufbau dieses Prototyps. Die verwendete Hardware setzt sich aus einem Raspberry Pi 4 mit 4 GB Arbeitsspeicher, einer Pi Camera V3 und einem MacBook Pro mit M1-Chipsatz zusammen. Der Raspberry Pi übernimmt dabei die Funktion eines Streaming-Servers. Die Kamera ist mit dem Raspberry Pi (Server) verbunden und sendet die Daten über das Netzwerk an das MacBook (Client). Auf dem MacBook läuft ein Webbrowser, der den Livestream empfängt und wiedergibt. Die nachfolgenden Abschnitte bieten eine detaillierte Betrachtung der eingesetzten Hardware- und Softwarekomponenten sowie ihrer jeweiligen Rollen im Streaming-Prozess.

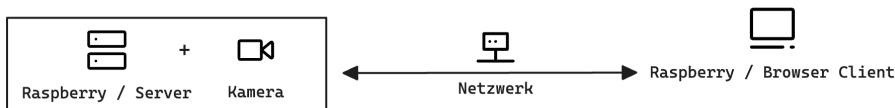


Abbildung 4.4: Schematische Darstellung des Streaming-Prototyps

4.3.1 Hardware Setup

Der Server für den Streaming Prototypen basiert auf einem Raspberry Pi 4. Dieser wurde wegen seiner H.264-Hardware-Encodierung und der für Streamingzwecke angemessenen Rechenleistung ausgewählt. Diese Leistungsfähigkeit der Hardware ist mit dieser welche potentiell in einem Anwendungsszenarien verbaut wird, mit der des Raspberry Pi 4 vergleichbar.

- Der Raspberry Pi 4 verfügt über einen Broadcom BCM2711 Quad-Core Cortex-A72 (ARM v8) 64-Bit SoC mit 1,8 GHz.
- Verschiedene Modelle mit 1 GB bis 8 GB LPDDR4-3200 SDRAM sind verfügbar, hier wird das 4 GB Modell verwendet.
- Er bietet Dual-Band IEEE 802.11ac WLAN und Bluetooth 5.0 sowie einen Gigabit-Ethernet-Anschluss für Netzwerke.
- Anschlüsse umfassen zwei USB 3.0- und zwei USB 2.0-Ports, sowie einen 40-poligen GPIO-Header.
- Für die Videoausgabe gibt es zwei Micro-HDMI®-Ports (bis zu 4K bei 60 Hz) und zusätzliche Anschlüsse für Display und Kamera.
- Er unterstützt H.265 (4kp60 Dekodierung), H264 (1080p60 Dekodierung, 1080p30 Kodierung) sowie OpenGL ES 3.1 und Vulkan 1.0.
- Ein Micro-SD-Kartenslot dient zur Datenspeicherung und dem Laden des Betriebssystems.

Als Bildaufnahmegerät wird die Pi Camera v3 eingesetzt, die sich über den MIPI CSI-Kameraport mit dem Raspberry Pi 4 verbindet. Diese Kamera ist für die Aufzeichnung von visuellen Daten konzipiert und bietet dank ihrer hohen Auflösung und vielseitigen Funktionen eine ausgezeichnete Bildqualität [Ltd23].

- **Auflösung:** Die Kamera bietet 11,9 Megapixel und kann Standbilder mit 4608×2592 Pixeln aufnehmen.
- **Video-Modi** Sie unterstützt diverse Formate wie 1080p50, 720p100 und 480p120.

Die Netzwerkanbindung erfolgt über Gigabit-Ethernet. Als Netzwerk-Switch wird eine UniFi Dream Machine von Ubiquiti Inc. verwendet, die eine Bandbreitenbeschränkung ermöglicht. Diese Switch unterscheidet sich in ihrer Funktionsweise nicht wesentlich von anderen handelsüblichen Netzwerk-Switches. Als Client-Gerät in dieser Studie dient ein Laptop mit der Apple M1 CPU und 8GB Arbeitsspeicher.

4 Web-based Streaming

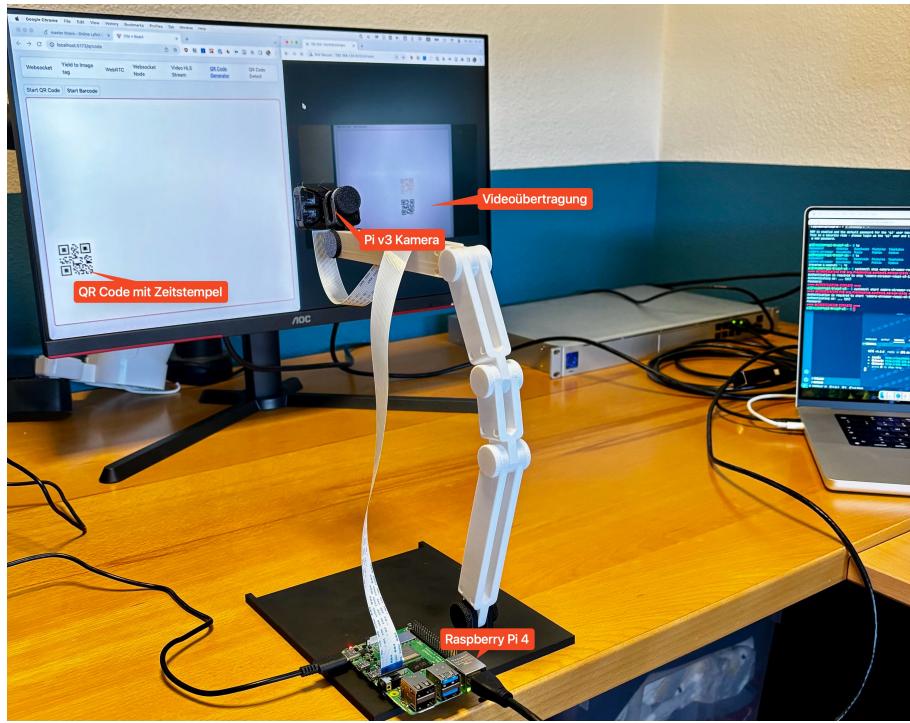


Abbildung 4.5: Setup eines aktiven Video Streams

4.3.2 Software Set Up

Wie in Abschnitt 4.3 gezeigt, erfolgt die Übertragung des Streams von einem Raspberry Pi, der als Server dient, zu einem Client im Browser. Die Backend-Implementierung ist unter <https://github.com/vbrantner/thesis-backend> zugänglich. Ebenso ist die Frontend-Implementierung unter <https://github.com/vbrantner/ma-thesis-frontend> einsehbar. Zusätzlich zum Testen von WebRTC wird das Tool *camera-streamer* eingesetzt⁸. Dieses Tool lässt sich auf Linux-Systemen installieren und ermöglicht die Erstellung eines MJPEG- oder WebRTC-Streams mit einer Kamera. Der Stream kann anschließend im Browser aufgerufen werden.

4.3.3 Backend

Das Backend basiert auf einem Linux-System. Für das Streaming kommen verschiedene Technologien und Programmiersprachen zum Einsatz, darunter ffmpeg, Python und JavaScript.

Für den Streaming-Prototyp wird folgende Software im Backend verwendet:

- **Libcamera Python:** Einbindung der Kamera.
- **OpenCV:** Verwendung der Kamera und Bildverarbeitung.

⁸Ein Open-Source-Tool für Webbrowser-Video-Streaming, verfügbar unter <https://github.com/ayufan/camera-streamer>

- **Websockets:** Eine Bibliothek zum Aufbau von WebSocket-Servern.
- **Camera-Streamer:** Eine Softwarelösung für das Streaming von Kamerabildern (<https://github.com/ayufan/camera-streamer>).
- **Flask:** Ein Web-Framework für Python.
- **FFMPEG:** Ein Programm zur Bearbeitung und Umwandlung von Multimedia-Dateien.
- **Linux Distribution Debian Bookworm:** Die verwendete Linux-Distribution.
- **DEBIAN Linux:** Das Betriebssystem für das Backend.

4.3.4 Frontend

Das Frontend nutzt folgende Technologien:

- **React:** Eine JavaScript-Bibliothek für den Aufbau von Benutzeroberflächen.
- **Vite:** Ein modernes Frontend-Tooling-System für schnelles Bundling und Entwicklung.
- **JSMPEG:** Eine JavaScript-Bibliothek für das Streamen von Videointhalten.

Die Webanwendung wurde unter Verwendung von React.js entwickelt, um eine dynamische und interaktive Benutzeroberfläche zu ermöglichen. Um das Streaming zu starten, muss zunächst das Backend aktiviert werden. Dazu kann die Datei `single_images_yield.py` im thesis-backend Repository gestartet werden. Anschließend ist es notwendig, die korrekte Netzwerkadresse im Frontend einzufügen, nämlich in folgender Datei: <https://github.com/vbrantner/ma-thesis-frontend/blob/main/src/routes/singleImagesStream.jsx>. Des Weiteren muss das Frontend gestartet werden.

4.4 Vergleich

In diesem Abschnitt werden die einzelnen Technologien welche fürs Streaming Möglich wären anhand der definierten Kriterien in 4.1 verglichen:

Geringe Latenz Die Latenz ist ein entscheidender Faktor. Technologien wie Einzelbilder, WebRTC und JSMPEG bieten bei ausreichender Bandbreite niedrige Latenzzeiten, was sie besonders für Echtzeitanwendungen geeignet macht. Im Gegensatz dazu weisen HLS und DASH aufgrund ihres segmentbasierten (buffer) Streamings eine höhere Latenz (> 1 Sekunde) auf.

Implementierungsaufwand Der geringste Aufwand ist beim Versenden von Einzelbildern über einen simplen HTTP-Stream oder WebSocket-Transport erforderlich. JSMPEG, DASH und HLS benötigen einen höheren Aufwand, wobei Bibliotheken von Drittanbietern die Implementierung erleichtern können. WebRTC verlangt den größten Implementierungsaufwand, da es ursprünglich für Peer-to-Peer-Verbindungen konzipiert war, obwohl vollständige Server-Implementierungen wie ⁹ und Ustreamer ¹⁰ zur Verfügung stehen.

Implementierungsaufwand Der geringste Aufwand ist beim Versenden von Einzelbildern über einen simplen HTTP-Stream oder WebSocket-Transport erforderlich. JSMPEG, DASH und HLS benötigen einen höheren Aufwand, wobei Bibliotheken von Drittanbietern die Implementierung erleichtern können. WebRTC verlangt den größten Implementierungsaufwand, da es ursprünglich für Peer-to-Peer-Verbindungen konzipiert war, obwohl vollständige Server-Implementierungen wie ¹¹ und Ustreamer ¹² zur Verfügung stehen.

Unterstützung Mehrerer Clients Alle Technologien ermöglichen das Streaming an mehrere Clients. Einzelbilder sind jedoch wegen der damit verbundenen hohen Bandbreitenanforderungen weniger geeignet. WebRTC, HLS und DASH zeigen gute Skalierbarkeit und Eignung für viele parallele Verbindungen.

Systemzuverlässigkeit In Bezug auf die Zuverlässigkeit des Systems zeichnen sich WebRTC und HLS durch ihre Robustheit und Langzeitstabilität aus. Sie bieten auch Qualitätsanpassungen (QoS), die die Bitrate oder Auflösung je nach Netzwerkbedingungen anpassen können.

Encoding Das Encoding ist ein wesentlicher Aspekt des Streamings. Moderne Codecs wie H.264 und VP9 werden von den meisten Technologien unterstützt, wobei speziell WebRTC und HLS für ihre effiziente Codierung und geringen Qualitätsverluste bekannt sind.

Ressourcen Die Nutzung von Ressourcen variiert stark zwischen den Technologien. Während WebRTC und HLS relativ ressourcenintensiv sind, insbesondere bei der Skalierung auf viele Clients, sind einfache HTTP-Streams bei Einzelbildern weniger anspruchsvoll.

Bandbreite Die Bandbreitenanforderungen sind ein kritischer Faktor. Einzelbilder verbrauchen viel Bandbreite, was ihre Skalierbarkeit begrenzt. WebRTC und HLS sind effizienter im Umgang mit der Bandbreite und passen ihre Streaming-Qualität dynamisch an die Netzwerkkapazität an.

Zur übersichtlichen Darstellung der Bewertung jeder Technologie habe ich eine Tabelle erstellt, die mit "+", "Oöder" die einzelnen Kriterien bewertet. Die Tabelle kann in 4.1 eingesehen werden, um einen detaillierten Vergleich der Streaming-Technologien zu ermöglichen.

⁹<https://github.com/pion/webrtc>

¹⁰Ustreamer

¹¹<https://github.com/pion/webrtc>

¹²Ustreamer

Kriterien / Technologie	Einzelbilder	WebRTC	HLS / DASH	JSMPEG
Geringe Latenz	+: Echtzeitübertragung möglich, abhängig vom Netzwerk.	+: Geringe verzögerung, geeignet für Echtzeitübertragung	-: Latenz durch buffer Streaming, Latenz >1s	+: Echtzeitübertragung möglich
Implementierung	+: sehr einfach im Backend, sowie Frontend	-: sehr Komplexe Implementierung	0: Hohe Anzahl an Bibliotheken verfügbar	-: Aufwendige Implementierung
Anzahl Clients	0: möglich, Skalierung schlecht	+: Für Anwendungsfall ausreichend	+: Skaliert gut für viele Nutzer.	+: Für Anwendungsfall ausreichend
Zuverlässigkeit	0: schwankungen im Netzwerk können Probleme verursachen	+: Implementiertes QoS	+: Sehr stabil und zuverlässig	-
Browserkomp.	+: sehr gute Unterstützung	+: sehr gute Unterstützung moderne Browser	+: Kompatibel mit modernen Webbrowsern	-: Basiert auf Open Source Projekt
Qualität	+: JPEG Verlustfrei oder auch Verlustfreiformate	-: H264 ist z.B. ein verlustbehaf tetest Format	-: H264 ist ein verlustbehaf tetes Format	-: MPEG1-Kompression führt zu Qualitätseinbußen.
Ressourcen	0: moderat, sehr gering, wenn Hardware Encoding möglich ist	0: Hardware Encoding sollte vorausgesetzt werden	0: Hardware Encoding sollte vorausgesetzt werden	0: Moderat
Bandbreite	-: sehr hohe Bandbreiten Nutzung	+: Geringe Bandbreitennutzung	+: Geringe Bandbreitennutzung	0: Mittel im Vergleich

Tabelle 4.1: Vergleich verschiedener Streaming-Technologien anhand ausgewählter Kriterien.

5 Analyse von web-based streaming

In diesem Kapitel erfolgt eine ausführliche Auswertung verschiedener webbasierter Streaming-Technologien. Zu Beginn werden die für die Bewertung herangezogenen Metriken definiert und erläutert. Anschließend wird die Methodik beschrieben. Die daraus resultierenden Ergebnisse werden präsentiert und diskutiert, um letztlich eine fundierte Beurteilung der Leistungsfähigkeit und Eignung der betrachteten Streaming-Technologien zu ermöglichen.

5.1 Metriken

Dieser Abschnitt widmet sich der Begründung und Erläuterung der ausgewählten Metriken, die für die Bewertung der Streaming-Qualität essenziell sind. Die betrachteten Metriken umfassen:

- **Latenz:** Diese Metrik misst die Zeitverzögerung zwischen dem Senden und Empfangen eines Bildes. Die Latenz ist ein kritischer Faktor für die Beurteilung der Effizienz einer Streaming-Technologie.
- **Jitter:** Hierbei wird die Variabilität der Latenz über einen definierten Zeitraum analysiert. Jitter-Indikatoren sind entscheidend für die Bewertung der Stabilität und Konsistenz der Datenübertragung.
- **Speichernutzung:** Diese Metrik untersucht, inwieweit das Streaming den Speicherplatz beeinflusst. Dabei wird zwischen der Nutzung auf dem Server und dem Client differenziert.
- **CPU-Auslastung:** Diese Metrik erfasst die Intensität der CPU-Belastung durch das Streaming. Auch hier erfolgt eine getrennte Betrachtung für Server und Client, um ein umfassendes Bild der Ressourcennutzung zu erhalten.
- **Bandbreite:** Untersucht wird wie hoch die Bandbreite ist während des Video Streaming.

5.2 Methodik

Das Ziel dieser Analyse besteht darin, verschiedene Streaming-Technologien – speziell Einzelbilder, JPEG, WebRTC und JSJPEG – gemäß festgelegter Kriterien zu evaluieren. Die Auswahl dieser Technologien basiert auf ihrer Fähigkeit, Streaming mit Latzenzen unter 200 Millisekunden zu unterstützen und eine reibungslose Darstellung in Webbrowsern zu gewährleisten. Der Grund für die Festlegung auf 200 Millisekunden ergibt sich aus den Erkenntnissen der Literaturrecherche in Kapitel 3.1.

5 Analyse von web-based streaming

Für die Datenerhebung wurden unterschiedliche Streaming-Lösungen unter variierenden Bedingungen getestet: verschiedene Auflösungen, kontrollierte Netzwerkbedingungen und die genannten Streaming-Technologien. Konkret erfolgte der Test durch ein 5-sekündiges Streaming, bei dem mittels Bildschirmaufnahme das Originalbild mit dem übertragenen Bild verglichen wird. Hierbei wurde ein QR-Code zur Erstellung eines Zeitstempels verwendet.

Die eingesetzte Streaming-Hardware und -Software wird in 4.3 beschrieben. Zusätzlich zur Streaming-Hardware wurde ein Bildschirm mit einer Bildwiederholrate von 240 Hz für die Anzeige verwendet. Als Browser kam Google Chrome zum Einsatz, in dem eine React-App für die Anzeige von QR-Codes genutzt wurde.

Die Bewertung orientierte sich an folgenden Kriterien:

- **Latenz:** Zeitdauer von der Bildübertragung bis zur Darstellung beim Empfänger.
- **Jitter:** Variation der Latenz über einen bestimmten Zeitraum.
- **Speichernutzung:** Auswirkungen des Streamings auf den Speicher des Servers.
- **CPU-Auslastung:** CPU-Belastung durch das Streaming, betrachtet für den Server.
- **Bandbreitenbeschränkung:** Einfluss einer limitierten Bandbreite auf das Streaming sowie Messung der Bandbreite während des Streams.

Die Datenauswertung erfolgte wie folgt:

1. Erzeugen eines QR-Codes mit Zeitstempel (Quelle).
2. Übertragung des Videos mittels gewählter Streaming-Technologie zu einem zweiten Browser (Ziel).
3. Gleichzeitige Aufnahme von Quelle und Ziel, beispielsweise durch Bildschirmaufnahme.
4. Zerlegung der Aufnahme in einzelne Frames.
5. Aufteilung jedes Frames in zwei Bilder (links und rechts) und deren Kennzeichnung.
6. Auslesen der QR-Codes für jede Bildhälfte und Berechnung der Zeitdifferenz.
7. Ermittlung von Maximal- und Minimalwerten zur Berechnung des Jitters.

Die Datenverarbeitung wird mit einem Python-Skript durchgeführt, das mittels Pandas die Durchschnittswerte, Maxima und Minima für den Jitter berechnet. Es ist zu beachten, dass dieser Ansatz voraussetzt, dass Quelle und Ziel des Streams an einer Stelle, wie beispielsweise einem Bildschirm, erfasst werden können. Andere in Abschnitt 3.3 beschriebene Ansätze nutzen synchronisierte Uhren, was notwendig ist, wenn keine direkte Aufnahme von Quelle und Ziel möglich ist. Während des Streams werden dann mit Hilfe von Linux-Kommandos wie top und systat die CPU- und Speicherauslastung sowie die Netzwerkbandbreite gemessen.

Abbildung 5.1 veranschaulicht schematisch das Setup für die Streaming-Messungen. Diese gesamte Auswertung wird für verschiedene Parameter sowie für die Technologien MJPEG, WebRTC und JSJPEG durchgeführt.

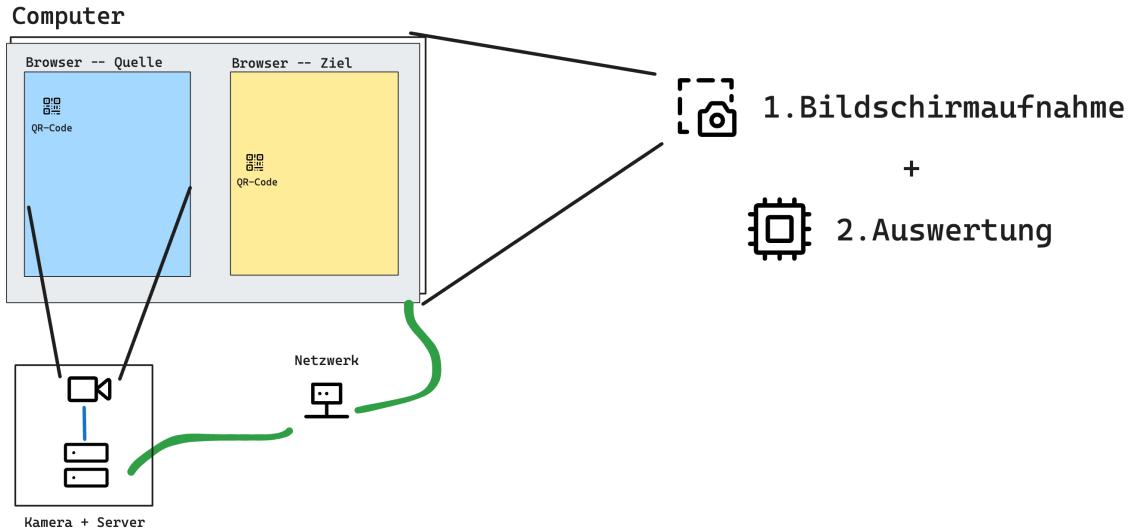


Abbildung 5.1: Bildschirmaufnahme der Übertragung des QR Codes

5.3 Implementierung

Die Implementierung zur Analyse webbasierter Streaming-Technologien besteht aus zwei Hauptkomponenten: Einer Web-App¹, die einen Timestamp in einen QR-Code umwandelt, und einem Python-Skript, das die Latenz aus Videoaufnahmen einer Streaming-Übertragung auswertet. Skripte für die Auswertung ist unter folgendem Link verfügbar <https://github.com/vbrantner/ma-latenerkennung>

Die Web-App unterstützt die Erstellung eines QR-Codes mit einem Zeitstempel, um so die Stream-Latenz messen zu können. In einem Intervall von 60 Sekunden wird der aktuelle Zeitstempel in einen QR-Code² umgewandelt. Die Position des QR-Codes ändert sich dabei kontinuierlich innerhalb eines Bereichs von 4x4 Feldern, um Überlappungen und Unlesbarkeit zu vermeiden. Abbildung 5.2 zeigt links die Web-App mit einem dargestellten QR-Code und rechts den abspielenden Video-Stream. Ein mögliches Problem bei der Aufnahme kann auftreten, wenn sich der QR-Code während der Belichtungszeit der Kamera ändert, was zur Darstellung zweier QR-Codes führen kann. Die dynamische Positionierung der QR-Codes hilft, derartige Probleme zu reduzieren.

Der zweite Teil der Implementierung, ein Python-Skript, berechnet aus einem Video die Latenz, den Durchschnittswert der Latenz sowie den Jitter über den gesamten Verlauf. Dazu wird eine Bildschirmaufnahme oder ein Video mit FFMPEG in Einzelbilder zerlegt. Der Befehl `ffmpeg -i "/path/video" -vf fps=60 ./images/frame%06d.png` erzeugt Bilder mit einer Frequenz von 60 Frames pro Sekunde aus dem Video.

Für die Netzwerkanalyse wurde das tool³ verwendet, mit diesem Tool ist es möglich die durchschnittliche Bandbreite Mbps zu ermitteln.

¹Web-App welche verwendet wird um QR Codes zu erzeugen <https://github.com/vbrantner/ma-thesis-frontend>

²QR Code Bibliothek für Node.js <https://github.com/soldair/node-qrcode>

³<https://github.com/sysstat/sysstat>

5 Analyse von web-based streaming

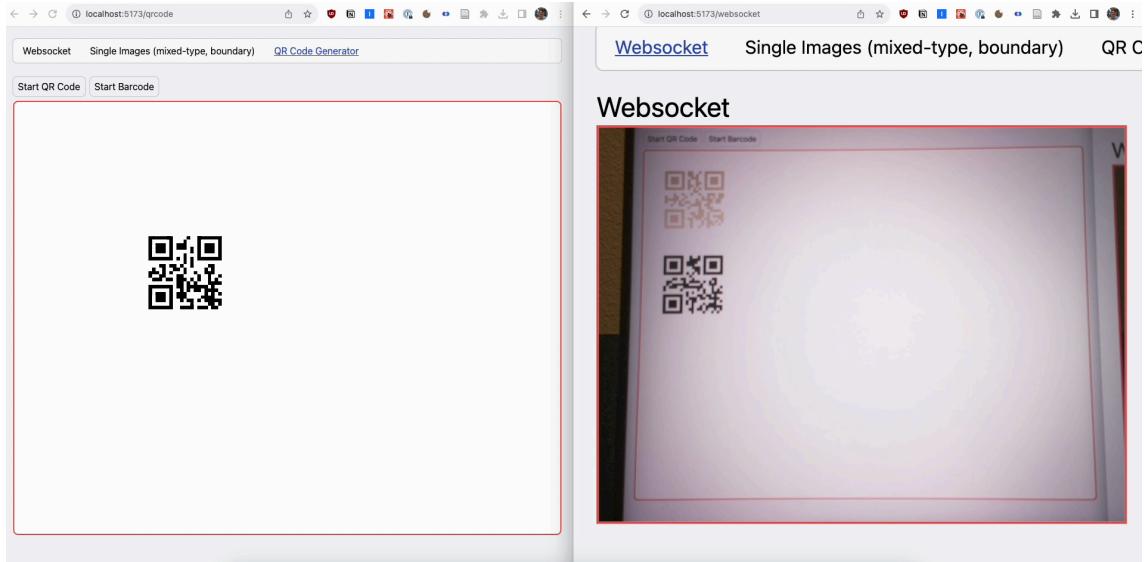


Abbildung 5.2: Bildschirmaufnahme der Übertragung des QR Codes

img_id	data_point
212	1691849653114
257	1691849652226
113	1691849651203
156	1691849651697
587	1691849654474
41	1691849647706

Tabelle 5.1: Timestamp-Werte, die aus den Bildern extrahiert wurden

Der top-Befehl oder auch "table of process" wird verwendet, um zu sehen, wie viel CPU und Speicher Programme verwenden. Es kann mit dem Toolset procsps⁴ installiert werden. Wenn man top ausführt zeigt es eine Liste von allen laufenden Programmen. In dieser Liste kann man sehen, wie viel Prozent der CPU und des Speichers jedes Programm benutzt. %CPU zeigt, wie viel von der CPU und %MEM zeigt, wie viel vom Speicher benutzt wird. Der top-Befehl aktualisiert diese Informationen regelmäßig, sodass man immer die neuesten Daten sieht.⁵

Jedes Frame erhält eine fortlaufende ID. Anschließend werden die Frames mittels des Befehls convert -crop 50%x100% "images/filename" "./split_images/with_id.png" in der Mitte geteilt, um zwei separate Bilder zu erhalten. Das Python-Skript verarbeitet die linke Hälfte aller geteilten Bilder, liest den Timestamp aus und speichert diesen zusammen mit der ID. Dasselbe Verfahren wird auf den rechten Bildteil angewandt. Tabelle 5.1 zeigt exemplarische Werte.

⁴<https://github.com/warmchang/procps>

⁵Mehr Infos unter <https://github.com/>

Jedes Frame erhält eine fortlaufende ID. Anschließend werden die Frames mittels des Befehls `convert -crop 50%x100% "images/filename" "./split_images/with_id.png"` in der Mitte geteilt, um zwei separate Bilder zu erhalten. Das Python-Skript verarbeitet die linke Hälfte aller geteilten Bilder, liest den Timestamp aus und speichert diesen zusammen mit der ID. Dasselbe Verfahren wird auf den rechten Bildteil angewandt. Tabelle 5.1 zeigt exemplarische Werte.

5.4 Ergebnisse

Das Ziel der Untersuchung ist, die Latenz, Bandbreite sowie CPU- und Speicherverbrauch der Streaming-Teilnehmer zu messen. Für die Übertragung kamen verschiedene Parameter zum Einsatz. In den Grafiken wurden folgenden Abkürzungen für die verschiedenen Streaming Technologien verwendet. Die Abkürzung CS steht für camera-streamer damit ist das Projekt ⁶ gemeint. Die Abkürzung öwnsteht für die eigene implementiert MJPEG Lösung mit Python. Getestet wurden folgende Faktoren:

- Auflösungen:
 - Niedrig: 1280x960
 - Hoch: 1600x1200
- Netzwerkbandbreite:
 - Hoch: Standard-Gigabit-Ethernet 1000 Mbps
 - Niedrig: 25 Mbps

Folgende Streaming-Technologien wurden verwendet mit jeweils 40 FPS:

- MJPEG (Camera-Streamer)
- WebRTC (Camera-Streamer)
- Einzelbildübertragung (eigene Implementierung)
- JSJPEG (GitHub Projekt) ⁷

Für jeden Parameter sowie Streaming Lösung wurde dann die CPU und Speicherauslastung sowie die durchschnittliche Bandbreite im Mbps aufgezeichnet.

⁶<https://github.com/ayufan/camera-streamer>

⁷Open Source MPEG1 Javascript Videoplayer <https://github.com/phoboslab/jsmpeg>

5.4.1 Latenz

Die Ergebnisse der Latenz bestehen aus den jeweils unterschiedlichen Auflösungen sowie den verschiedenen Technologien und den Netzwerk einschränkungen. Die Latenzzeiten wurden als Differenz zwischen zwei Zeitstempeln (in Millisekunden) für jedes übertragene Bild gemessen. Der Boxplot in Abbildung 5.3 veranschaulicht die Verteilung der Latenzzeiten über die verschiedenen Streaming-Techniken hinweg, wenn das Netzwerk nicht Eingeschränkt ist. Im folgenden die Ergebnisse:

- ***res_low_mjpeg_own*** zeigte eine durchschnittliche Latenzzeit von etwa -83 ms . Die Latenzzeiten variierten in einem relativ schmalen Bereich, was sich in einer Spanne von $25,25\text{ ms}$ zwischen dem unteren und oberen Quartil widerspiegelt. Die Extremwerte lagen zwischen -112 ms und -51 ms .
- ***res_low_cs_webrtc*** hatte einen Median von -109 ms und wies eine breitere Streuung auf, erkennbar an einer Spanne von $28,25\text{ ms}$ zwischen dem unteren und oberen Quartil. Die Latenz reichte von -147 ms bis -54 ms .
- ***res_low_cs_mjpeg*** zeigte ebenfalls einen Median von -83 ms , jedoch mit einer engeren Streuung, dargestellt durch eine Spanne von 6 ms zwischen den Quartilen. Die Extremwerte lagen bei -115 ms und -51 ms .
- ***jsmpeg*** zeigte die höchste durchschnittliche Latenz von -161 ms und eine breite Streuung mit einer Spanne von 30 ms zwischen den Quartilen. Die Latenzzeiten variierten zwischen -225 ms und -119 ms .
- ***res_high_own_mjpeg*** hatte einen Median von -139 ms und eine geringe Streuung, ersichtlich an einer Quartilspanne von 6 ms . Die Latenz reichte von -199 ms bis -87 ms .
- ***res_high_cs_webrtc*** zeigte eine mittlere Latenzzeit von -141 ms und eine Quartilspanne von $27,50\text{ ms}$, was auf eine mäßige Streuung hindeutet. Die Latenzzeiten variierten zwischen -199 ms und -107 ms .
- ***res_high_cs_mjpeg*** zeigte einen Median von -113 ms und eine breitere Streuung, erkennbar an einer Quartilspanne von 27 ms . Die Extremwerte lagen bei -166 ms und -80 ms .
- ***visionline*** hatte eine durchschnittliche Latenz von -86 ms und eine Quartilspanne von 29 ms , was auf eine mittlere Streuung hindeutet. Die Latenz reichte von -143 ms bis -51 ms .

Wenn jedoch das Netzwerk eingeschränkt ist zeigen die verschiedenen Streaming-Techniken *mjpeg_own*, *cs_webrtc* und *cs_mjpeg*.

- ***mjpeg_own*** zeigte eine durchschnittliche Latenzzeit von $-2103,94\text{ ms}$. Der Median lag bei $-2130,91\text{ ms}$, mit einer Standardabweichung von $149,24\text{ ms}$. Die Latenzzeiten variierten zwischen $-2356,36\text{ ms}$ und $-1800,45\text{ ms}$.
- ***cs_webrtc*** wies eine deutlich geringere durchschnittliche Latenzzeit von $-143,36\text{ ms}$ auf. Der Median lag hier bei -141 ms , mit einer Standardabweichung von $18,54\text{ ms}$. Die Latenzzeiten reichten von -201 ms bis -109 ms .

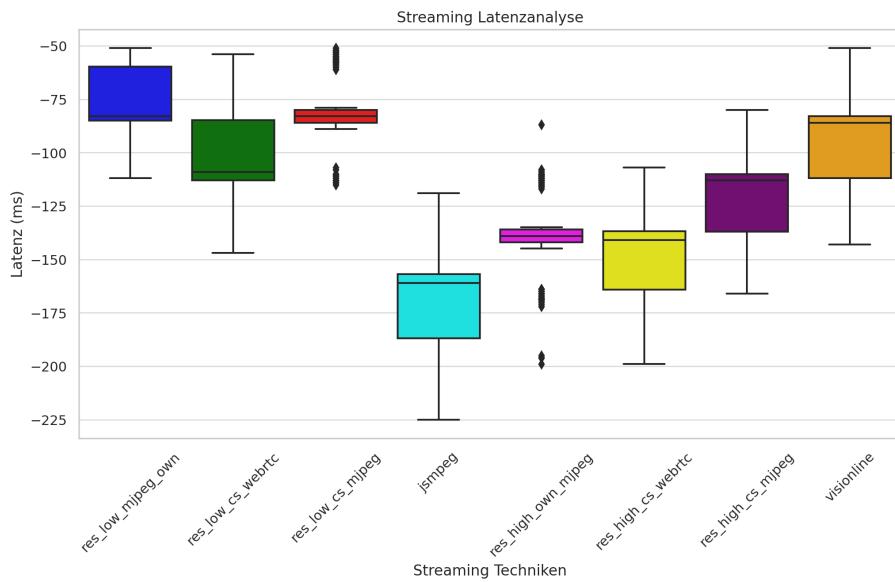


Abbildung 5.3: Unterschiedliche Streaming Latenzen, keine Netzwerkbandbreite Einschränkung

- *cs_mjpeg* hatte eine durchschnittliche Latenzzeit von $-2023,00\text{ ms}$, mit einem Median von -2041 ms . Die Standardabweichung betrug $254,34\text{ ms}$, und die Latenzzeiten lagen zwischen -2573 ms und -1374 ms .

Diese Ergebnisse sind im beigefügten Boxplot 5.4 visualisiert, der die Verteilung der Latenzzeiten für jede Technik zeigt. Jede Technik ist durch eine eigene Farbe dargestellt: *mjpeg_own* in Blau, *cs_webrtc* in Grün und *cs_mjpeg* in Rot.

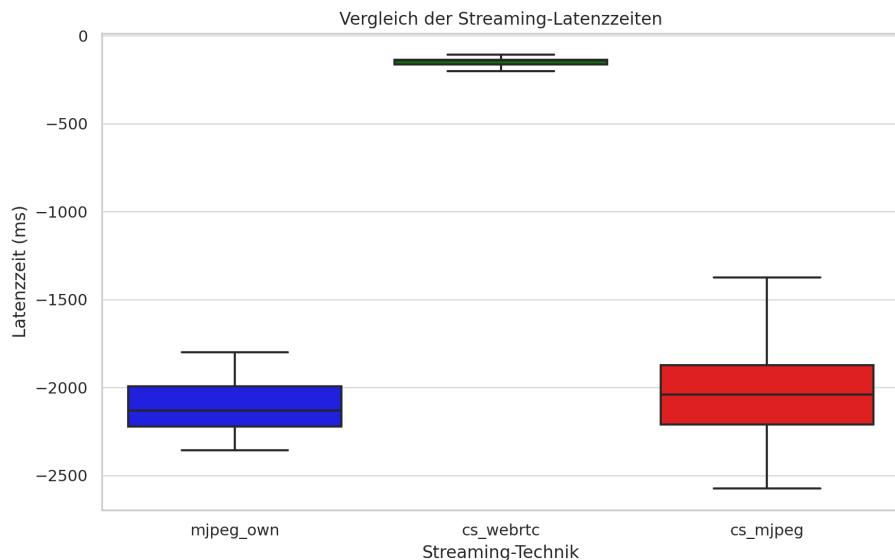


Abbildung 5.4: Boxplot der Streaming-Latenzzeiten, bei hoher Auflösung, niedriger Netzwerkbänderbreite

5 Analyse von web-based streaming

Die Analyse der Latenzzeiten für Streaming-Techniken, wenn die Bandbreite Eingeschränkt ist sowie die Auflösung niedrig ist zeigt der Boxplot 5.5 folgende Ergebnisse:

- **cs-webrtc** verzeichnete eine durchschnittliche Latenzzeit von $-105,12\text{ ms}$, mit einer Standardabweichung von $16,53\text{ ms}$. Der Median lag bei -110 ms . Die Latenzzeiten variierten zwischen -164 ms und -55 ms .
- **cs-mjpeg** wies eine durchschnittliche Latenzzeit von $-436,47\text{ ms}$ auf, mit einer Standardabweichung von $38,28\text{ ms}$. Der Median betrug -445 ms , und die Latenzzeiten lagen zwischen -530 ms und -335 ms .
- **own-mjpeg** hatte eine ähnliche durchschnittliche Latenzzeit von $-431,08\text{ ms}$, mit einer Standardabweichung von $44,75\text{ ms}$. Der Median lag bei -423 ms . Die Latenzzeiten reichten von -536 ms bis -309 ms .

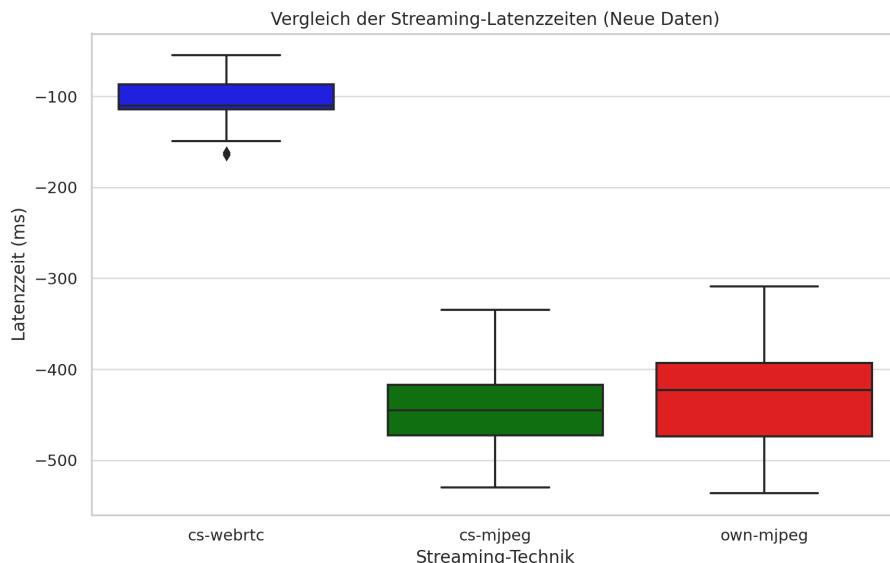


Abbildung 5.5: Boxplot der Streaming-Latzenzeiten (Neue Daten)

5.4.2 Netzwerkbandbreite

Dieser Teil präsentiert die genutzte Bandbreite unter den verschiedenen Testbedingungen. Diagramme und Tabellen veranschaulichen den Datenverbrauch der einzelnen Streaming-Technologien. Die Analyse der Daten aus verschiedenen Streaming-Techniken zeigt folgende Ergebnisse und werden in der Abbildung

1. **JSMPEG:** Diese Technik weist eine durchschnittliche Übertragungsrate (TX) von 5.13 Mbps auf.
2. **MJPEG (Hohe Auflösung):** Bei dieser Technik liegt die durchschnittliche TX-Rate bei 70.71 Mbps, was deutlich höher ist als bei den anderen Techniken.
3. **WebRTC (Hohe Auflösung):** Diese Technik zeigt eine durchschnittliche TX-Rate von 3.05 Mbps, was die niedrigste Rate unter den getesteten Hochauflösungstechniken ist.

4. **MJPEG (Niedrige Auflösung):** Die durchschnittliche TX-Rate für diese Technik beträgt 54.17 Mbps.
5. **WebRTC (Niedrige Auflösung):** Hier liegt der Durchschnitt bei 2.36 Mbps, was die niedrigste Rate unter allen getesteten Techniken darstellt.

Diese Ergebnisse zeigen, dass die Wahl der Streaming-Technik einen erheblichen Einfluss auf die benötigte Bandbreite hat, insbesondere in Bezug auf die Auflösung. MJPEG mit hoher Auflösung benötigt die höchste Bandbreite, während WebRTC in niedriger Auflösung am wenigsten Bandbreite beansprucht.

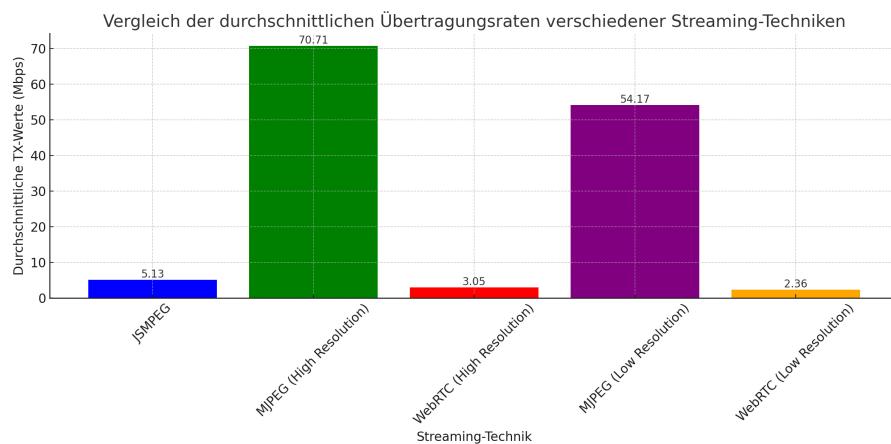


Abbildung 5.6: Vergleich der Bandbreiten Nutzung bei unterschiedlichen Einstellungen

5.4.3 CPU- und Speicherauslastung

In diesem Abschnitt wird der Verbrauch von CPU und Speicher für jede Streaming-Technologie aufgeführt. Die Ergebnisse werden durch Diagramme ergänzt, die die Auslastung unter verschiedenen Bedingungen zeigen.

Der Vergleich der Streaming-Techniken auf dem Raspberry Pi mit 4 GB RAM zeigt deutliche Unterschiede in der CPU- und Speichernutzung. Die folgende Analyse basiert auf Durchschnittswerten der gemessenen Daten.

JSMPEG zeigte eine moderate CPU-Auslastung und Speichernutzung, was auf eine effiziente Verarbeitung hinweist. Im Gegensatz dazu wies MJPEG in niedriger Auflösung (eigene Implementierung) eine höhere CPU-Last bei vergleichbarer Speichernutzung auf. WebRTC in niedriger Auflösung (CS) und hoher Auflösung (CS) demonstrierten eine ähnliche CPU-Nutzung, jedoch mit steigender Speicheranforderung bei höherer Auflösung.

MJPEG in hoher Auflösung (eigene Implementierung) beanspruchte erheblich mehr CPU-Ressourcen, was auf eine intensivere Verarbeitung hinweist. MJPEG in niedriger und hoher Auflösung (CS) zeigte ebenfalls eine erhöhte CPU-Auslastung, wobei die Speichernutzung in der höheren Auflösung geringfügig stieg.

5 Analyse von web-based streaming

Diese Ergebnisse zeigen, dass die Wahl der Streaming-Technik und deren Auflösung signifikante Auswirkungen auf die Leistungsfähigkeit des Raspberry Pi haben. Die Entscheidung für eine Technik sollte daher unter Berücksichtigung der spezifischen Anforderungen an CPU- und Speicherkapazität erfolgen.

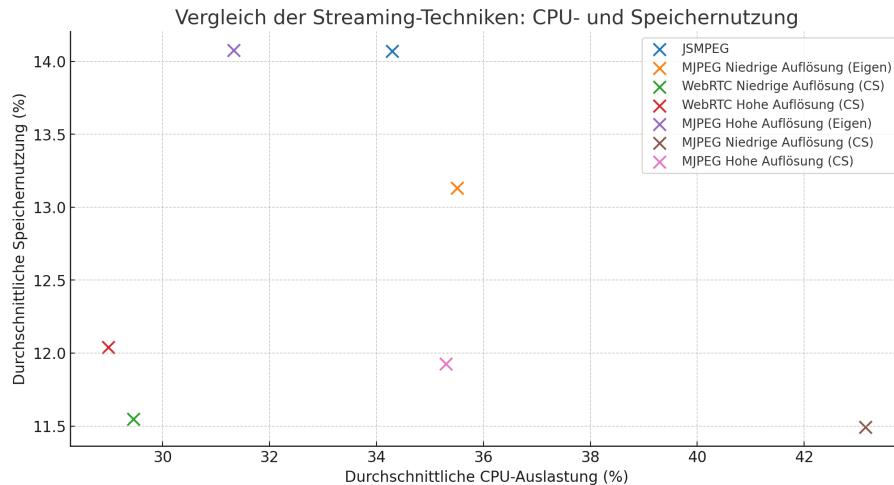


Abbildung 5.7: Vergleich der Speicher und CPU Auslastung in Prozent

5.5 Diskussion

In Abschnitt 4 wurde gezeigt, dass MJPEG, WebRTC sowie die Software JSMPEG für Video-streaming im Browser geeignet sind. Für eine detaillierte Analyse wurden diese vier Streaming-Technologien mit zwei verschiedenen Auflösungen sowie mit einer Einschränkung der Netzwerkbandbreite und ohne solche getestet. Zusammengefasst waren die folgenden Faktoren maßgeblich:

- Auflösungen:
 - Niedrig: 1280x960
 - Hoch: 1600x1200
- Netzwerkbandbreite:
 - Hoch: Standard-Gigabit-Ethernet 1000 Mbps
 - Niedrig: 25 Mbps

Unter Berücksichtigung der Latenz erzielte das Streaming mit MJPEG die besten Resultate. Bei hoher Auflösung konnte eine durchschnittliche Latenz von 113 ms erreicht werden, bei niedriger Auflösung lag die Latenz bei 83 ms. Ein Problem von MJPEG ist jedoch der sehr hohe Bandbreitenbedarf von bis zu 70 Mbit/s bei hoher Auflösung. Die CPU- und Speicherauslastung bewegen sich im Rahmen, da sie im getesteten Prozess nie 80% darüber hinaus ist. Der Implementationsaufwand für MJPEG relativ gering, da der Browser Bilder über das HTML-Element anzeigen kann und der Datentransfer im Browser über HTTP oder WebSocket erfolgen kann.

WebRTC liefert ebenfalls gute Latenzwerte. Bei hoher Auflösung lag die durchschnittliche Latenz bei 141 ms. Die Latenz ist bei WebRTC etwas höher als bei MJPEG, die Netzwerkbandbreite ist jedoch etwa 20 Mal niedriger und beträgt nur etwa 3 Mbit/s. Ein Nachteil von WebRTC ist die aufwändige Serverseitenimplementierung, da es keinen standardisierten Lösungsansatz gibt, um eine Videoübertragung vom Server zu einem Client zu ermöglichen. Hier können, wie in diesem Fall, Drittanbieter-Libraries wie camera-streamer verwendet werden oder für individuelle Lösungen, Libraries wie pion⁸.

JSMPEG konnte in den durchgeführten Tests leider nicht überzeugen. Selbst ohne Netzwerkeinschränkungen und bei niedriger Auflösung wurde nur eine Latenz von 160 ms erreicht. Gründe für diese unzureichende Leistung konnten nicht ermittelt werden. Es ist jedoch erwähnenswert, dass die CPU-Auslastung während der Ausführung von JSMPEG gelegentlich bedeutend erhöht war.

⁸Pion WebRTC ist ein Open-Source-Projekt, das eine WebRTC-Implementierung in der Programmiersprache Go bereitstellt. Es ermöglicht Echtzeitkommunikation und den Peer-to-Peer-Datenaustausch in Webanwendungen.

6 Augmentation im web-based Streaming

6.1 Design Studie

In der vorliegenden Masterarbeit wird ein Interviewverfahren eingesetzt, um Visualisierungstechniken für die Augmentation von Echtzeit-Videostreaming in Webbrowsern zu entwickeln. Das Hauptziel dieser Studie besteht darin, die technologischen Möglichkeiten und Herausforderungen bei der Integration von Augmented Reality (AR) in Echtzeit-Videostreams zu analysieren und die praktische Anwendbarkeit dieser Technologien aufzuzeigen. Die Methodik der Designstudie orientiert sich an dem von Sedlmair et al. [SMM12] vorgeschlagenen Ansatz, welcher die Durchführung einer Untersuchung in mehreren Phasen empfiehlt. Die einzelnen Phasen umfassen:

- Lernen: Eintauchen in das Fachgebiet, um ein tiefes Verständnis für das Problem zu entwickeln.
- Sichten: Identifikation relevanter Daten, Werkzeuge und Methoden für das Projekt.
- Zuordnen: Festlegung der spezifischen Aufgaben und Ziele der Studie.
- Entdecken: Entwicklung von Hypothesen und Ideen für Designlösungen.
- Entwerfen: Erstellung vorläufiger Designkonzepte und Prototypen.
- Implementieren: Entwicklung des endgültigen Visualisierungssystems.
- Bereitstellen: Einsatz des Systems in einer realen Umgebung.
- Reflektieren: Bewertung der Effektivität und des Einflusses des Systems.
- Schreiben: Dokumentation der Studie und ihrer Ergebnisse.

6.2 Methodik

Im Rahmen dieser Forschungsarbeit wird ein qualitatives Interview durchgeführt, um ein vertieftes Verständnis der Anforderungen und Herausforderungen im Bereich des Echtzeit-Videostreamings und der Augmented Reality bei Laserbearbeitungsprozessen zu gewinnen. Ziel ist es, die Forschungsfrage zu klären, wie ein Visualisierungswerkzeug optimal gestaltet werden kann, um den spezifischen Bedürfnissen der Stakeholder zu entsprechen.

Der Interviewprozess orientiert sich an einem semi-strukturierten Leitfaden, der Flexibilität in den Antworten ermöglicht und gleichzeitig sicherstellt, dass alle relevanten Themenbereiche abgedeckt werden. Die Auswahl des Interviewpartners erfolgt basierend auf seiner Expertise im Bereich der Datenvisualisierung und seiner Relevanz für das Forschungsthema. Die Methodik der Datenerhebung und -analyse lehnt sich an die „Design Study Methodology“ von Sedlmair und Munzner an, wird

6 Augmentation im web-based Streaming

jedoch an die spezifischen Bedingungen dieser Studie angepasst. Dies beinhaltet eine detaillierte thematische Analyse der Interviewdaten, um Muster und spezifische Anforderungen für das zu entwickelnde Visualisierungswerkzeug zu identifizieren.

Die Ergebnisse des Interviews fließen direkt in die Konzeption und Entwicklung des Visualisierungswerkzeugs ein, wobei technische Aspekte und Benutzerfreundlichkeit besonders berücksichtigt werden. Die Methodik zielt darauf ab, ein Werkzeug zu entwickeln, das eng an den realen Bedürfnissen der Zielgruppe ausgerichtet ist. Dabei werden die Limitationen der Studie, insbesondere die Beschränkung auf ein einzelnes Interview, kritisch reflektiert und Strategien zur Validierung der Ergebnisse berücksichtigt.

In der abschließenden Phase wird das Interview ausgewertet, und es werden mögliche Szenarien für die Visualisierung entwickelt. Diese Methodik stellt somit einen pragmatischen und zielgerichteten Forschungsansatz dar, der sowohl die theoretischen als auch die praktischen Aspekte der Thematik berücksichtigt.

6.3 Interview & Ergebnisse

Das im Rahmen dieser Studie durchgeführte Interview folgte einem semi-strukturierten Ansatz, um eine flexible und zugleich fokussierte Diskussion zu ermöglichen. Dieser Ansatz erlaubte es dem Befragten, seine Erfahrungen und Einsichten frei zu teilen, während gleichzeitig spezifische Themenbereiche, die für diese Forschung relevant sind, abgedeckt wurden. Der im Anhang beigefügte Fragenkatalog diente als Grundgerüst für das Interview, ermöglichte aber auch spontane Nachfragen und thematische Abweichungen, um tiefere Einblicke in spezifische Aspekte des Forschungsthemas zu gewinnen. Diese Methodik erwies sich als besonders effektiv, um detaillierte und praxisnahe Informationen über die Anwendung und Entwicklung von Technologien im Bereich der Mikro-Material-Bearbeitung zu sammeln.

6.3.1 Persona

Der Befragte ist promoviert im Fachbereich Elektrotechnik und arbeitet als Entwicklungsingenieur für Mikro-Material-Bearbeitung. Sein Tätigkeitsbereich umfasst die Entwicklung neuer Technologien für den Einsatz von Ultrakurzpulslasern in langfristigen Projekten sowie die Erforschung neuer Anwendungsfelder mit einem Fokus auf Mikrolaserbearbeitung.

6.3.2 VisionLine-Produkte und Positionsausrichtung

Der Befragte erwähnt Kundenanforderungen für VisionLine-ähnliche Lösungen zur Erkennung und Ausrichtung von Bearbeitungsstrukturen. Es besteht Potenzial für innovative Visualisierungen, die dem Nutzer helfen könnten, Positionierungen präziser und effizienter durchzuführen. Anwendungen in einer Augmented-Reality-Umgebung könnten die Genauigkeit bei der Platzierung und Bearbeitung von Materialien verbessern. Der Befragte illustriert dies am Beispiel eines Kunden, der eine Anlage für spezifische Materialbearbeitungen nutzt und dabei auf solche Visualisierungstechniken angewiesen ist. [...] Also sind wir eher nicht bei den großen Produktionsstraßen, sondern nur bei dem einen Kunden, der sich mal eine Anlage irgendwie reinstellt und der hat dann ein Material,

was vielleicht eine Struktur hat, die er auch mal drehen möchte und möchte von vorne und von hinten ungefähr an der gleichen Stelle bearbeiten. Und der hätte dann gerne solche VisionLine-Detect-Lösungen, wo er dann sein Loch erkennen kann und man von dem irgendwo seine Gravur ausrichten kann [...] [...] Ich könnte mir prinzipiell vorstellen, dass wenn man ein sehr, sehr gutes Vision-System hätte, man deutlich günstigere Achsen einsetzen könnte, weil man prinzipiell mit dem Scanner die Genauigkeit erreichen könnte, wenn man genau wüsste, wo der steht und das direkt zurückführen könnte und das mit entsprechender Dynamik möglicherweise sogar zurückführen könnte [...]

6.3.3 Automatisierte Sensorik und Bildgebungsintegration

Laut dem Befragten könnten durch den Einsatz hochauflösender und präziser Sensoriksysteme weniger präzise und damit kostengünstigere mechanische Systeme eingesetzt werden. Dies würde die Verwendung fortgeschrittener Visualisierungstechniken zur Verbesserung der Genauigkeit von Bearbeitungsprozessen implizieren. Ein solches System könnte es ermöglichen, die Genauigkeit eines Scanners zu nutzen, um mechanische Unzulänglichkeiten auszugleichen.

6.3.4 Stitching-Technik und Navigationsoptimierung

Es werden Situationen erörtert, in denen große Werkstücke zusammengesetzt werden müssen. Das Stitching könnte automatisch korrigiert werden, um eine präzise Passform aller Teile eines zusammengesetzten Objekts zu gewährleisten. Die Navigation könnte durch solche Visualisierungstechniken erheblich verbessert werden, insbesondere in verzerrten Bereichen. Die Möglichkeit, Werkstücke zu stitchen, deutet darauf hin, dass Techniken wie Augmented Reality zur Verbesserung der Orientierung und Visualisierung im Arbeitsbereich beitragen könnten. *Wenn ich jetzt aber hier eben so einen Ausschnitt mit meinem Vision-System habe, dann sehe ich das teilweise schon auch. Wenn ich jetzt hier hinten ein Kamerabild mehr angucke und in der Mitte ein Kamerabild mehr angucke, dann ist es außen häufig verzerrt und innen eigentlich nicht gut. [...] Genau, und dann wäre halt die Idee, theoretisch das Werkstück komplett zu stitchen, damit man das einfach besser navigieren könnte.*

6.3.5 Echtzeit-Steuerung und präzise Pulsplatzierung

Die genaue Kenntnis darüber, an welcher Stelle welcher Puls platziert wird, unterstreicht die Notwendigkeit einer präzisen Steuerung. Möglicherweise könnte eine visuelle Augmentierung die Präzision verbessern und Fehlerquoten reduzieren. Der Einsatz von Puls-Lasern mit sehr hohen Repetitionsraten erfordert eine entsprechend leistungsfähige Steuerungstechnologie, um die exakte Platzierung jedes Pulses sicherzustellen.

6.4 AR Grundlagen

Für eine effektive Demonstration des 3D-CAD-Overlays über einem realen Objekt wird eine markerbasierte Erkennungstechnologie eingesetzt. Der Hauptgrund für diese Wahl ist, dass OpenCV, eine weit verbreitete Computer Vision Bibliothek, eine spezialisierte API anbietet. Diese API ermöglicht die Erkennung von Markern sowie die Bestimmung der Pose, was essentiell ist, um beispielsweise in einem Video-Stream kontinuierlich die exakte Rotation und Position in Bezug zur Kamera zu erfassen [Ope23]. Neben dem Marker-Tracking existieren auch andere Technologien wie das Linien-Tracking und Feature-Tracking, die ebenfalls in bestimmten Anwendungsfällen relevant sein können [NO23].

6.5 AR Visualisierung

Die Implementierung der AR Visualisierung in dieser Thesis konzentriert sich auf die Überlagerung eines CAD-Modells über ein reales Objekt, um so Informationen wie zukünftige Arbeitsschritte im Schweißprozess einer Lasermaschine zu visualisieren. Dies beinhaltet Angaben über Form und Intensität des Schweißvorgangs, was für Überprüfungszwecke von Bedeutung ist.

In einem ersten Schritt wurde ein markerbasiertes Tracking mit AR.js und A-Frame implementiert. AR.js basiert auf ARToolkit, einer Open-Source-Bibliothek zur Erstellung von Augmented-Reality-Anwendungen. A-Frame ist ein Web-Framework zur Erstellung von Virtual-Reality-Erlebnissen. Dieser Ansatz nutzte einfaches Tracking mittels eines Markers, stieß jedoch auf technische Einschränkungen. Die Verwendung von AR.js und A-Frame ermöglichte nur den Einsatz von Webkameras als Videoquelle, was die Anwendungsmöglichkeiten einschränkte. AR.js¹ und A-Frame².

Um eine größere Flexibilität und erweiterte Funktionen zu erreichen, wurde als nächstes eine Kombination aus OpenCV und Three.js implementiert. OpenCV, eine umfangreiche Open-Source-Bibliothek für Computer Vision, wurde in Verbindung mit Aruco-Markern genutzt³. Three.js, ein JavaScript-Framework, kam zum Einsatz, um 3D-Elemente im Frontend darzustellen. Diese Kombination ermöglichte eine verbesserte Pose-Estimation und erweiterte die Möglichkeiten der Kamerasteuerung über die Einschränkungen einer reinen Webkamera-Anbindung hinaus. OpenCV⁴ und Three.js⁵.

Die Herausforderung bei der Implementierung mit OpenCV und Three.js lag in der technischen Realisierung der Pose-Estimation und der nahtlosen Integration der Kamera-Daten in das Three.js-Framework. Diese Hürde konnte nicht vollständig überwunden werden, was zeigt, dass weitere

¹AR.js ist eine JavaScript-Bibliothek für Augmented Reality Anwendungen im Web, die auf ARToolkit basiert: <https://ar-js-org.github.io/AR.js-Docs/>

²A-Frame ist ein Web-Framework für die Erstellung von 3D- und VR-Erlebnissen: <https://aframe.io/>

³Der Code für das Backend kann unter <https://github.com/vbrantner/augmentation-backend> eingesehen werden

⁴OpenCV ist eine Open-Source-Computer-Vision-Bibliothek, die vielfältige Algorithmen für Bildverarbeitung und Maschinelles Sehen bietet: <https://opencv.org/>

⁵Three.js ist ein JavaScript-Framework zur Erstellung und Anzeige von 3D-Grafiken im Webbrowser: <https://threejs.org/>

Listing 6.1 Python-Code zur Echtzeit-Erkennung von ArUco-Markern in einem Videostream und zur Schätzung ihrer Pose mit OpenCV.

```
# Start von Video Input
cap = cv2.VideoCapture(0)
cap.set(3, 800)
cap.set(4, 600)

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()
    if not ret:
        break

    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detect ArUco markers
    corners, ids, rejectedImgPoints = aruco.detectMarkers(
        gray, aruco_dict, parameters=parameters
    )

    # If markers are detected
    if len(corners) > 0:
        # Draw the markers on the frame
        frame = aruco.drawDetectedMarkers(frame, corners, ids)

        # Estimate pose of each marker and return the values rvec and tvec
        rvecs, tvecs, _ = aruco.estimatePoseSingleMarkers(
            corners, 0.05, camera_matrix, distortion_coefficients
        )
        for rvec, tvec in zip(rvecs, tvecs):
            aruco.drawAxis(
                frame, camera_matrix, distortion_coefficients, rvec, tvec, 0.03
            )

    rotation_matrix = cv2.Rodrigues(rvecs[0])[0]
    euler_angles = rotationMatrixToEulerAngles(rotation_matrix)
    ...
```

Forschung und Entwicklung in diesem Bereich erforderlich sind, um eine optimale Lösung zu finden. Die gewonnenen Erkenntnisse bieten jedoch eine solide Grundlage für zukünftige Verbesserungen und Anpassungen in der AR-Visualisierung.

6 Augmentation im web-based Streaming



Abbildung 6.1: CAD 3D Overlay, welche mit realem Objekt übereinstimmt

6.6 Diskussion

Um eine schnellere Untersuchung zu ermöglichen, wurde für die Designstudie ein verkürzter methodischer Ansatz gewählt, der sich hauptsächlich auf Interviews und eine Demonstrationsimplementierung stützte. Der Verzicht auf einen umfassenderen und vielschichtigeren Forschungsansatz, wie etwa eine ausgedehnte Auswahl von Stakeholdern oder einen iterativen Designprozess in den Fragen sowie in der Analyse der Interviewergebnisse, führte dazu, dass nur bedingt Einblicke gewonnen werden konnten.

Das durchgeführte Interview lieferte dennoch Einblicke in die potenziellen Anwendungen von Augmented Reality (AR) im Echtzeit-Videostreaming. Ein Beispiel hierfür ist die Visualisierung von 3D-Modellen innerhalb des Streams, um die Navigation und operationale Klarheit zu verbessern. Zusätzlich wurde die Möglichkeit hervorgehoben, zukünftige Laserbearbeitungsprozesse zu visualisieren, was es den Benutzern ermöglicht, die Aktionen der Lasermaschine vorherzusehen und zu verstehen. Diese Einblicke unterstreichen den praktischen Nutzen und die benutzerzentrierten Vorteile der Integration von AR in solche Systeme.

Bei dem Versuch, eine AR-Visualisierung zu implementieren, traten technische Herausforderungen auf. Die Integration von OpenCV für das Backend-Tracking und Three.js für das Frontend-3D-Modellieren konnte nicht erfolgreich umgesetzt werden.

Die genaue Abbildung der Kameraposition von OpenCV auf die Kameraperspektive oder -position im Three.js-Umfeld stellte sich als schwierig heraus. In Abbildung 6.1 sieht man, dass die Abstimmung ziemlich gut passt. In Abbildung 6.2 wird das virtuelle Modell nicht richtig positioniert.

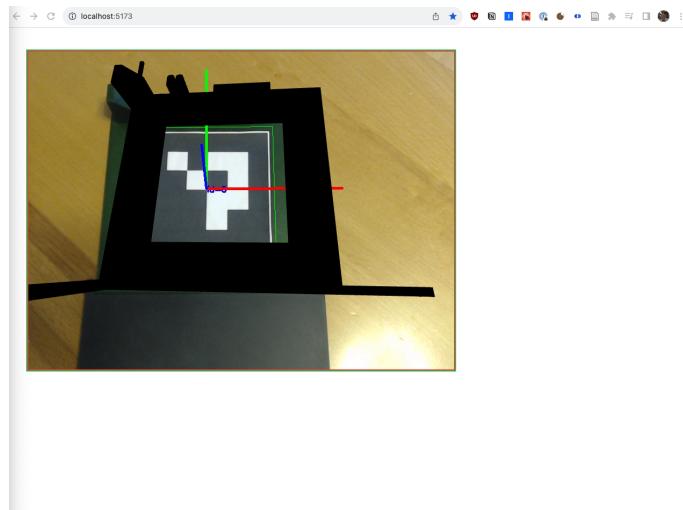


Abbildung 6.2: CAD 3D Overlay mit OpenCV und Three.js

Trotz der aufgetretenen Herausforderungen stellt Three.js⁶, eine beliebte Bibliothek im JavaScript-Ökosystem, ein vielversprechendes Werkzeug für das webbasierte 3D-Modellieren dar. Seine weite Verbreitung und Vielseitigkeit legen nahe, dass es mit weiterer Entwicklung und Verfeinerung, insbesondere im Kontext von AR-Anwendungen, eine praktikable Lösung für zukünftige Forschungs- und Implementierungsbemühungen sein könnte.

⁶Three.js, eine führende 3D JavaScript-Bibliothek, hat auf GitHub über 90k Sterne erlangt. <https://github.com/mrdoob/three.js>

7 Zusammenfassung

7.1 Zusammenfassung und Schlussfolgerungen

Diese Arbeit stellt vor, wie ein Video-Stream mit niedriger Latenz im Webbrowser implementiert werden kann, um eine Echtzeit-Interaktion zu ermöglichen. Es wurde eine umfassende Analyse der Technologien MJPEG und WebRTC durchgeführt, um deren Eignung für Streaming-Anwendungen mit niedriger Latenz zu bewerten. Die Ergebnisse sind wie folgt zusammengefasst:

MJPEG

- Latenz bei einer Auflösung von 1600x1200 und 40 fps: 113 ms
- Bandbreite: 70 Mbit/s
- Bei einer Auflösung von 1280x960: Latenz von 83 ms
- Höhere Speicher- und CPU-Auslastung im Vergleich zu WebRTC
- Kann bei Wi-Fi-Clients und bei mehreren gleichzeitigen Empfängern zu Netzwerkproblemen führen

WebRTC

- Latenz bei einer Auflösung von 1600x1200: 140 ms
- Bandbreite: ca. 3 Mbit/s
- Bei einer Auflösung von 1280x960: Latenz von 109 ms
- Geringere Speicher- und CPU-Auslastung als MJPEG
- Bessere Eignung für hohe Auflösungen und moderne Videokodierung
- Komplexere Serverseitige Implementierung erforderlich, verwendung zum Beispiel von Pion WebRTC¹ möglich

¹Pion WebRTC ist eine Open-Source WebRTC-Implementierung in Go, die Echtzeitkommunikation in Webanwendungen ermöglicht.

7 Zusammenfassung

Ein Vergleich mit der aktuellen Streaming-Lösung von TRUMPF Laser, die nicht auf Webtechnologie basiert, zeigte eine Latenz von 86 ms. Es wird deutlich, dass beide Technologien MJPEG und WebRTC ihre spezifischen Vor- und Nachteile haben. MJPEG bietet eine geringere Latenz, leidet aber unter hohem Bandbreitenbedarf und höherer Ressourcennutzung. WebRTC bietet eine effizientere Nutzung der Bandbreite und Ressourcen, erfordert jedoch eine komplexere Implementierung, insbesondere serverseitig.

Darüber hinaus wurde die Möglichkeit erörtert, Video-Streams mit digitalen Informationen zu erweitern, insbesondere durch die Integration und Entwicklung von 3D-Modellen im Browser. Hierbei kann die Verwendung von React und Bibliotheken wie Three.js Vorteile bieten, indem sie die Erstellung von 3D-Modellen im Browser vereinfachen.

Zusammenfassend lässt sich feststellen, dass die Wahl der Streaming-Technologie stark von den spezifischen Anforderungen der Anwendung abhängt. Während MJPEG für Szenarien mit niedrigerer Auflösung und geringerer Bandbreitenbeschränkung geeignet ist, bietet WebRTC Vorteile bei höheren Auflösungen und in Umgebungen, in denen eine effiziente Bandbreitennutzung entscheidend ist. Beide Technologien haben das Potenzial, Echtzeit-Interaktionen im Webbrowser zu ermöglichen, wobei die Entscheidung für eine Technologie von den spezifischen Anforderungen und Rahmenbedingungen der jeweiligen Anwendung abhängt.

7.2 Implikationen

Es ist zu beachten, dass die Design-Studie in einem begrenzten Rahmen stattfand. Es wurde lediglich ein Interview ausgewertet, ohne eine umfassende Stakeholder-Analyse oder einen interaktiven Ansatz, der beispielsweise eine Demo der Visualisierung und ein nachfolgendes Interview beinhaltet.

Bei der Messung der Latenz für WebRTC wurde auf eine Drittanbieter-Bibliothek zurückgegriffen. Dies bedeutet, dass die gemessenen Latenzwerte bei einer eigenen Implementierung abweichen könnten, obwohl sie voraussichtlich in einem ähnlichen Bereich liegen würden. Es ist wichtig, dies bei der Interpretation der Ergebnisse zu berücksichtigen.

Ein weiterer Aspekt, der in der Studie nicht detailliert behandelt wurde, ist die Aufschlüsselung der Latenz in ihre verschiedenen Komponenten. Es wäre von Vorteil, genau zu verstehen, welchen Anteil die Kamera, das Kodieren, das Übertragen der Daten, das mögliche Dekodieren im Browser und die Darstellung im Browser selbst an der Gesamtlatenz haben. Ein detaillierteres Verständnis dieser Komponenten könnte zu effizienteren Optimierungen in Zukunft führen.

7.3 Zukünftige Arbeit

Im Bereich der Videostream-Technologien sind, Stand 2023, zwei Technologien sehr vielversprechend, die das Streaming von Videoinhalten vom Server zum Browser standardisieren und verbessern. Diese beiden Technologien sind die WebCodecs API [Moz23g] und WebTransport in Verbindung mit HTTP/3 und QUIC [Moz23i].

Derzeit besteht eine Herausforderung beim Video-Encoding und -Decoding darin, dass diese Prozesse entweder mit WebAssembly (WASM) oder mit JavaScript (JS) eigenständig entwickelt werden müssen. Die WebCodecs API zielt darauf ab, eine standardisierte Schnittstelle bereitzustellen, die eine optimierte Nutzung von Hardware-Ressourcen ermöglicht, abhängig von der Implementierung des jeweiligen Browsers [Moz23g].

WebTransport hingegen nutzt die Vorteile von QUIC, einer Alternative zu TCP [Luk23]. QUIC integriert Merkmale von UDP und bietet eine kontrollierte Überlastungssteuerung, was eine effizientere Datenübertragung ermöglicht. Das Projekt "quic.video"² demonstriert die Anwendung dieser Technologie.

Ein weiteres Feld für zukünftige Arbeit bietet die korrekte Augmentation von 3D-Overlays, basierend auf Marker-Tracking sowie auch auf nicht-Marker-Tracking. Obwohl dies ein lösbares technisches Problem darstellt, könnten in Zukunft Verfahren von Interesse sein, die auf nicht-markerbasiertem Tracking aufbauen. Solche Ansätze könnten zusätzliche Flexibilität und Anwendungsmöglichkeiten in verschiedenen Kontexten bieten, indem sie die Abhängigkeit von physischen Markern reduzieren und somit eine nahtlose Integration von virtuellen und realen Elementen ermöglichen.

²Ein Video Player für ein Backend sowie eine Javascript-Bibliothek für die Einbindung im Frontend <https://quic.video/source/>

Literaturverzeichnis

- [Ama23] Amazon Web Services, Inc. *Webanwendungen vs. Native Apps vs. Hybrid Apps – Arten von Anwendungen im Vergleich – AWS*. Amazon Web Services, Inc. 2023. URL: <https://aws.amazon.com/de/compare/the-difference-between-web-apps-native-apps-and-hybrid-apps/> (besucht am 30. 10. 2023) (zitiert auf S. 21).
- [App23] Apple Inc. *HTTP Live Streaming*. Apple Developer Documentation. 2023. URL: <https://developer.apple.com/documentation/http-live-streaming> (besucht am 30. 10. 2023) (zitiert auf S. 17, 48).
- [Azu97] R. T. Azuma. „A Survey of Augmented Reality“. In: *Presence: Teleoperators and Virtual Environments* 6.4 (Aug. 1997), S. 355–385. ISSN: 1054-7460. doi: [10.1162/pres.1997.6.4.355](https://doi.org/10.1162/pres.1997.6.4.355). URL: <https://direct.mit.edu/pvar/article/6/4/355-385/18336> (besucht am 05. 12. 2023) (zitiert auf S. 28).
- [Bac19] C. Bachhuber. „Low Delay Video Communication“. In: (2019). (Besucht am 30. 10. 2023) (zitiert auf S. 31, 32).
- [BF11] T. Berners-Lee, M. Fischetti. *Weaving the web: the original design and ultimate destiny of the World Wide Web by its inventor*. 1. paperback ed., [repr.] New York, NY: HarperBusiness, 2011. 246 S. ISBN: 978-0-06-251587-2. (Besucht am 30. 10. 2023) (zitiert auf S. 21).
- [Bin15] B. Bing. *Next-generation video coding and streaming*. Hoboken, New Jersey: Wiley, 2015. 321 S. ISBN: 978-1-118-89130-8. (Besucht am 30. 10. 2023) (zitiert auf S. 27, 28).
- [BSTC17] Y. Bandung, L. B. Subekti, D. Tanjung, C. Chrysostomou. „QoS analysis for Web-RTC videoconference on bandwidth-limited network“. In: *2017 20th International Symposium on Wireless Personal Multimedia Communications (WPMC)*. 2017 20th International Symposium on Wireless Personal Multimedia Communications (WPMC). Bali: IEEE, Dez. 2017, S. 547–553. ISBN: 978-1-5386-2768-6. doi: [10.1109/WPMC.2017.8301873](https://doi.org/10.1109/WPMC.2017.8301873). URL: <http://ieeexplore.ieee.org/document/8301873/> (besucht am 07. 11. 2023) (zitiert auf S. 45).
- [Can23] Can I Use Github Community. "webrtc Can I use... Support tables for HTML5, CSS3, etc. Okt. 2023. URL: <https://caniuse.com/?search=webrtc> (besucht am 30. 10. 2023) (zitiert auf S. 43).
- [Chr23] Chromium Project. *Network Stack Chromium*. 2023. URL: <https://www.chromium.org/developers/design-documents/network-stack/> (besucht am 23. 09. 2023) (zitiert auf S. 23).
- [Clo23] Cloudflare Inc. *Why is HTTP not secure? | HTTP vs. HTTPS*. Cloudflare. 2023. URL: <https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/> (besucht am 23. 09. 2023) (zitiert auf S. 23).

- [DAE+20] K. Durak, M. N. Akcay, Y. K. Erinc, B. Pekel, A. C. Begen. „Evaluating the Performance of Apple’s Low-Latency HLS“. In: *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*. 2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP). Tampere, Finland: IEEE, 21. Sep. 2020, S. 1–6. ISBN: 978-1-72819-320-5. doi: [10.1109/MMSP48831.2020.9287117](https://doi.org/10.1109/MMSP48831.2020.9287117). URL: <https://ieeexplore.ieee.org/document/9287117/> (besucht am 17. 10. 2023) (zitiert auf S. 32).
- [Dom23] Dominic Szablewski. *JSMpeg – Decode it like it’s 1999*. Okt. 2023. URL: <https://jsmpeg.com/> (besucht am 30. 10. 2023) (zitiert auf S. 47).
- [Ecm23] Ecma International. *ECMAScript® 2024 Language Specification*. 2023. (Besucht am 30. 10. 2023) (zitiert auf S. 26).
- [ELI18] A. E. Essaili, T. Lohmar, M. Ibrahim. „Realization and Evaluation of an End-to-End Low Latency Live DASH System“. In: *2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*. 2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB). Valencia: IEEE, Juni 2018, S. 1–5. ISBN: 978-1-5386-4729-5. doi: [10.1109/BMSB.2018.8436922](https://doi.org/10.1109/BMSB.2018.8436922). URL: <https://ieeexplore.ieee.org/document/8436922/> (besucht am 30. 10. 2023) (zitiert auf S. 32).
- [ETS23] O. El Marai, T. Taleb, J. Song. „AR-Based Remote Command and Control Service: Self-Driving Vehicles Use Case“. In: *IEEE Network* 37.3 (Mai 2023), S. 170–177. ISSN: 0890-8044, 1558-156X. doi: [10.1109/MNET.119.2200058](https://doi.org/10.1109/MNET.119.2200058). URL: <https://ieeexplore.ieee.org/document/9910350/> (besucht am 15. 10. 2023) (zitiert auf S. 31, 32).
- [GG06] A. Grosskurth, M. W. Godfrey. „Architecture and evolution of the modern web browser“. In: (2006). (Besucht am 30. 10. 2023) (zitiert auf S. 21–23).
- [Goo18a] Google LLC. *Inside look at modern web browser (part 1)*. Chrome for Developers. 5. Sep. 2018. URL: <https://developer.chrome.com/blog/inside-browser-part1/> (besucht am 23. 09. 2023) (zitiert auf S. 21, 22).
- [Goo18b] Google LLC. *Inside look at modern web browser (part 3)*. Chrome for Developers. 20. Sep. 2018. URL: <https://developer.chrome.com/blog/inside-browser-part3/> (besucht am 23. 09. 2023) (zitiert auf S. 21, 22).
- [Goo18c] Google LLC. *Inside look at modern web browser (part 4)*. Chrome for Developers. 21. Sep. 2018. URL: <https://developer.chrome.com/blog/inside-browser-part4/> (besucht am 23. 09. 2023) (zitiert auf S. 21, 23).
- [Gri13] I. Grigorik. *High-performance browser networking*. OCLC: ocn827951729. Beijing ; Sebastopol, CA: O’Reilly, 2013. 382 S. ISBN: 978-1-4493-4476-4. (Besucht am 30. 10. 2023) (zitiert auf S. 25).
- [GW18] R. C. Gonzalez, R. E. Woods. *Digital image processing*. New York, NY: Pearson, 2018. 1168 S. ISBN: 978-0-13-335672-4. (Besucht am 30. 10. 2023) (zitiert auf S. 27).

- [HMH+09] R. Hill, C. Madden, A. V. D. Hengel, H. Detmold, A. Dick. „Measuring Latency for Video Surveillance Systems“. In: *2009 Digital Image Computing: Techniques and Applications*. 2009 Digital Image Computing: Techniques and Applications. Melbourne, Australia: IEEE, 2009, S. 89–95. ISBN: 978-1-4244-5297-2. doi: [10.1109/DICTA.2009.23](https://doi.org/10.1109/DICTA.2009.23). URL: <http://ieeexplore.ieee.org/document/5384976/> (besucht am 23. 10. 2023) (zitiert auf S. 34).
- [Inf22] InfoQ.com C4Media Inc. *JavaScript Reaches the Final Frontier: Space*. InfoQ. 2022. URL: <https://www.infoq.com/news/2020/06/javascript-spacex-dragon/> (besucht am 05. 09. 2023) (zitiert auf S. 18, 21).
- [JB13] J. Jansen, D. C. A. Bulterman. „User-centric video delay measurements“. In: *Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. MMSys ’13: Multimedia Systems Conference 2013. Oslo Norway: ACM, 26. Feb. 2013, S. 37–42. ISBN: 978-1-4503-1892-1. doi: [10.1145/2460782.2460789](https://doi.org/10.1145/2460782.2460789). URL: <https://dl.acm.org/doi/10.1145/2460782.2460789> (besucht am 23. 10. 2023) (zitiert auf S. 34).
- [JFF16] U. Jennehag, S. Forsstrom, F. Fiordigigli. „Low Delay Video Streaming on the Internet of Things Using Raspberry Pi“. In: *Electronics* 5.4 (20. Sep. 2016), S. 60. ISSN: 2079-9292. doi: [10.3390/electronics5030060](https://doi.org/10.3390/electronics5030060). URL: <http://www.mdpi.com/2079-9292/5/3/60> (besucht am 18. 10. 2023) (zitiert auf S. 35).
- [KAN13] A. Kryczka, A. Arefin, K. Nahrstedt. „AvCloak: A Tool for Black Box Latency Measurements in Video Conferencing Applications“. In: *2013 IEEE International Symposium on Multimedia*. 2013 IEEE International Symposium on Multimedia (ISM). Anaheim, CA, USA: IEEE, Dez. 2013, S. 271–278. ISBN: 978-1-4799-2171-3 978-0-7695-5140-1. doi: [10.1109/ISM.2013.52](https://doi.org/10.1109/ISM.2013.52). URL: <http://ieeexplore.ieee.org/document/6746804/> (besucht am 18. 10. 2023) (zitiert auf S. 34).
- [KP20] M. Koppehel, T. Pionteck. „Ultra-Low-Latency Video Encoding on Heterogenous Hardware Platforms“. In: *2020 International Conference on Field-Programmable Technology (ICFPT)*. 2020 International Conference on Field-Programmable Technology (ICFPT). Maui, HI, USA: IEEE, Dez. 2020, S. 287–287. ISBN: 978-1-66542-302-1. doi: [10.1109/ICFPT51103.2020.90049](https://doi.org/10.1109/ICFPT51103.2020.90049). URL: <https://ieeexplore.ieee.org/document/9415555/> (besucht am 15. 10. 2023) (zitiert auf S. 33).
- [LBR+23] T. Lyko, M. Broadbent, N. Race, M. Nilsson, P. Farrow, S. Appleby. „Improving quality of experience in adaptive low latency live streaming“. In: *Multimedia Tools and Applications* (12. Juli 2023). ISSN: 1380-7501, 1573-7721. doi: [10.1007/s11042-023-15895-9](https://doi.org/10.1007/s11042-023-15895-9). URL: <https://link.springer.com/10.1007/s11042-023-15895-9> (besucht am 15. 10. 2023) (zitiert auf S. 33).
- [Ltd23] R. P. Ltd. *Buy a Raspberry Pi Camera Module 3*. Raspberry Pi. 2023. URL: <https://www.raspberrypi.com/products/camera-module-3/> (besucht am 30. 10. 2023) (zitiert auf S. 49).
- [Luk23] Luke Curley. *Replacing WebRTC - Media over QUIC*. 2023. URL: <https://quic.video/blog/replacing-webrtc/> (besucht am 30. 10. 2023) (zitiert auf S. 77).
- [MDN23] MDN contributors. *HTML: HyperText Markup Language | MDN*. 17. Juli 2023. URL: <https://developer.mozilla.org/en-US/docs/Web/HTML> (besucht am 22. 09. 2023) (zitiert auf S. 21).

Literaturverzeichnis

- [Moz23a] Mozilla Corporation. *WebAssembly Concepts - WebAssembly | MDN*. 31. Mai 2023. URL: <https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts> (besucht am 23. 09. 2023) (zitiert auf S. 22).
- [Moz23b] Mozilla Foundation. *Fetch API - Web APIs | MDN*. 1. Apr. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (besucht am 14. 10. 2023) (zitiert auf S. 24).
- [Moz23c] Mozilla Foundation. *Image file type and format guide - Web media technologies | MDN*. 23. Nov. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Image_types (besucht am 30. 10. 2023) (zitiert auf S. 42).
- [Moz23d] Mozilla Foundation. *Introduction to web APIs - Learn web development | MDN*. 22. Juli 2023. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction (besucht am 14. 10. 2023) (zitiert auf S. 23).
- [Moz23e] Mozilla Foundation. *The WebSocket API (WebSockets) - Web APIs | MDN*. 22. Nov. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket_API (besucht am 30. 10. 2023) (zitiert auf S. 41).
- [Moz23f] Mozilla Foundation. *Web APIs | MDN*. 20. Feb. 2023. URL: <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest> (besucht am 14. 10. 2023) (zitiert auf S. 23).
- [Moz23g] Mozilla Foundation. *WebCodecs API - Web APIs | MDN*. 25. Okt. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebCodecs_API (besucht am 30. 10. 2023) (zitiert auf S. 25, 76, 77).
- [Moz23h] Mozilla Foundation. *WebRTC API - Web APIs | MDN*. 6. Nov. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API (besucht am 30. 10. 2023) (zitiert auf S. 41).
- [Moz23i] Mozilla Foundation. *WebTransport API - Web APIs | MDN*. 25. Okt. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebTransport_API (besucht am 30. 10. 2023) (zitiert auf S. 76).
- [NO23] A. Y. C. Nee, S. K. Ong, Hrsg. *Springer Handbook of Augmented Reality*. Springer Handbooks. Cham: Springer International Publishing, 2023. ISBN: 978-3-030-67821-0 978-3-030-67822-7. DOI: [10.1007/978-3-030-67822-7](https://doi.org/10.1007/978-3-030-67822-7). URL: <https://link.springer.com/10.1007/978-3-030-67822-7> (besucht am 28. 11. 2023) (zitiert auf S. 28, 70).
- [off21] office365itpros. *Teams 2.0 Promises Better Client Performance*. Section: Teams. 25. Juni 2021. URL: <https://office365itpros.com/2021/06/25/teams-2-webview2-replaces-electron/> (besucht am 30. 10. 2023) (zitiert auf S. 17).
- [Ope23] OpenCV. *OpenCV: Detection of ArUco Markers*. 2023. URL: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html (besucht am 30. 10. 2023) (zitiert auf S. 70).
- [ROGL18] L. Rodriguez-Gil, P. Orduña, J. García-Zubia, D. López-de-Ipiña. „Interactive live-streaming technologies and approaches for web-based applications“. In: *Multimedia Tools and Applications* 77.6 (März 2018), S. 6471–6502. ISSN: 1380-7501, 1573-7721. DOI: [10.1007/s11042-017-4556-6](https://doi.org/10.1007/s11042-017-4556-6). URL: <http://link.springer.com/10.1007/s11042-017-4556-6> (besucht am 15. 10. 2023) (zitiert auf S. 33, 41).

- [SCH+20] L. Song, D. Chen, T. Hou, T. Wang, B. Liu. „Automatic Docking System of Fuel Filler with CAD model-based tracking and Visual Servoing Control“. In: *2020 39th Chinese Control Conference (CCC)*. 2020 39th Chinese Control Conference (CCC). Shenyang, China: IEEE, Juli 2020, S. 6013–6017. ISBN: 978-988-15639-0-3. doi: [10.23919/CCC50068.2020.9189413](https://doi.org/10.23919/CCC50068.2020.9189413). URL: <https://ieeexplore.ieee.org/document/9189413/> (besucht am 30. 10. 2023) (zitiert auf S. 38).
- [Sev15] C. Severance. „Guido van Rossum: The Early Years of Python“. In: *Computer* 48.2 (Feb. 2015), S. 7–9. ISSN: 0018-9162, 1558-0814. doi: [10.1109/MC.2015.45](https://doi.org/10.1109/MC.2015.45). URL: <https://ieeexplore.ieee.org/document/7042702/> (besucht am 24. 09. 2023) (zitiert auf S. 27).
- [SMM12] M. Sedlmair, M. Meyer, T. Munzner. „Design Study Methodology: Reflections from the Trenches and the Stacks“. In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (Dez. 2012), S. 2431–2440. ISSN: 1077-2626. doi: [10.1109/TVCG.2012.213](https://doi.org/10.1109/TVCG.2012.213). URL: <http://ieeexplore.ieee.org/document/6327248/> (besucht am 30. 10. 2023) (zitiert auf S. 67).
- [SPL+20] D. Silhavy, S. Pham, M. Lasak, A. Chen, S. Arbanowski. „Low latency streaming and multi DRM with dash.js“. In: *Proceedings of the 11th ACM Multimedia Systems Conference*. MMSys ’20: 11th ACM Multimedia Systems Conference. Istanbul Turkey: ACM, 27. Mai 2020, S. 291–296. ISBN: 978-1-4503-6845-2. doi: [10.1145/3339825.3394936](https://doi.org/10.1145/3339825.3394936). URL: <https://dl.acm.org/doi/10.1145/3339825.3394936> (besucht am 15. 10. 2023) (zitiert auf S. 33).
- [SSP15] B. Sredojev, D. Samardzija, D. Posarac. „WebRTC technology overview and signaling solution design and implementation“. In: *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). Opatija, Croatia: IEEE, Mai 2015, S. 1006–1009. ISBN: 978-953-233-082-3. doi: [10.1109/MIPRO.2015.7160422](https://doi.org/10.1109/MIPRO.2015.7160422). URL: <http://ieeexplore.ieee.org/document/7160422/> (besucht am 07. 11. 2023) (zitiert auf S. 44).
- [Sta23] Stack Overflow. *Stack Overflow Developer Survey 2023*. Stack Overflow. 2023. URL: https://survey.stackoverflow.co/2023/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2023 (besucht am 24. 09. 2023) (zitiert auf S. 17, 26, 27).
- [Sto11] T. Stockhammer. „Dynamic adaptive streaming over HTTP –: standards and design principles“. In: *Proceedings of the second annual ACM conference on Multimedia systems*. MMSYS ’11: MMSYS ’11 - Multimedia Systems Conference. San Jose CA USA: ACM, 23. Feb. 2011, S. 133–144. ISBN: 978-1-4503-0518-1. doi: [10.1145/1943552.1943572](https://doi.org/10.1145/1943552.1943572). URL: <https://dl.acm.org/doi/10.1145/1943552.1943572> (besucht am 08. 11. 2023) (zitiert auf S. 47).
- [SZM16] Shuai Zhao, Zhu Li, D. Medhi. „Low delay MPEG DASH streaming over the WebRTC data channel“. In: *2016 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. 2016 IEEE International Conference on Multimedia & Expo Workshops (ICMEW). Seattle, WA, USA: IEEE, Juli 2016, S. 1–6. ISBN: 978-1-5090-1552-8. doi: [10.1109/ICMEW.2016.7574765](https://doi.org/10.1109/ICMEW.2016.7574765). URL: <http://ieeexplore.ieee.org/document/7574765/> (besucht am 30. 10. 2023) (zitiert auf S. 17, 33, 48).

Literaturverzeichnis

- [TLV08] M. Ter Louw, J. S. Lim, V. N. Venkatakrishnan. „Enhancing web browser security against malware extensions“. In: *Journal in Computer Virology* 4.3 (Aug. 2008), S. 179–195. ISSN: 1772-9890, 1772-9904. DOI: [10.1007/s11416-007-0078-5](https://doi.org/10.1007/s11416-007-0078-5). URL: <http://link.springer.com/10.1007/s11416-007-0078-5> (besucht am 21.09.2023) (zitiert auf S. 21).
- [TRU23a] TRUMPF SE + Co. KG. *Bildverarbeitung für Schneid- und Schweißanwendungen*. 2023. URL: https://www.trumpf.com/de_CH/produkte/laser/sensorik/bildverarbeitung-schneid-schweissanwendung/ (besucht am 30.10.2023) (zitiert auf S. 18).
- [TRU23b] TRUMPF SE + Co. KG. *Company profile*. 2023. URL: https://www.trumpf.com/en_US/company/profile/company-profile/ (besucht am 30.10.2023) (zitiert auf S. 18).
- [TRU23c] TRUMPF SE + Co. KG. *Kleiner, kleiner, immer feiner*. 2023. URL: https://www.trumpf.com/de_CH/produkte/laser/euv-drive-laser/ (besucht am 30.10.2023) (zitiert auf S. 18).
- [UP21] S. Ubik, J. Pospisilik. „Video Camera Latency Analysis and Measurement“. In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.1 (Jan. 2021), S. 140–147. ISSN: 1051-8215, 1558-2205. DOI: [10.1109/TCSVT.2020.2978057](https://doi.org/10.1109/TCSVT.2020.2978057). URL: <https://ieeexplore.ieee.org/document/9022890/> (besucht am 18.10.2023) (zitiert auf S. 36).
- [Vis23] Visometry GmbH. *VisionLib Demo Companion App – VisionLib*. 2023. URL: <https://visionlib.com/develop/resources/companionapp/> (besucht am 30.10.2023) (zitiert auf S. 29).
- [W3C23] W3C Community Group. *WebAssembly*. 2023. URL: <https://webassembly.org/> (besucht am 30.10.2023) (zitiert auf S. 26).
- [WEW+16] H. Wuest, T. Engekle, F. Wientapper, F. Schmitt, J. Keil. „From CAD to 3D Tracking — Enhancing & Scaling Model-Based Tracking for Industrial Appliances“. In: *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)*. 2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct). Merida, Yucatan, Mexico: IEEE, Sep. 2016, S. 346–347. ISBN: 978-1-5090-3740-7. DOI: [10.1109/ISMAR-Adjunct.2016.0114](https://doi.org/10.1109/ISMAR-Adjunct.2016.0114). URL: [http://ieeexplore.ieee.org/document/7836535/](https://ieeexplore.ieee.org/document/7836535/) (besucht am 27.11.2023) (zitiert auf S. 38).
- [Wor23] World Wide Web Consortium. *Discover W3C groups*. W3C. 2023. URL: <https://www.w3.org/groups/> (besucht am 14.08.2023) (zitiert auf S. 24).
- [WS07] H. Wuest, D. Stricker. „Tracking of industrial objects by using CAD models“. In: 4.1 (2007). (Besucht am 30.10.2023) (zitiert auf S. 37, 38).
- [YCK14] G. J. Yang, B. W. Choi, J. H. Kim. „Implementation of HTTP Live Streaming for an IP Camera using an Open Source Multimedia Converter“. In: *International Journal of Software Engineering and Its Applications* (2014). (Besucht am 30.10.2023) (zitiert auf S. 32).

Alle URLs wurden zuletzt am 25.11.2023 geprüft.

Interview Fragen

Zielgruppe

Frage: Welche Rolle, Aufgaben und Bereiche übernehmen Sie bei TRUMPF?

Frage: Können Sie Ihre berufliche Laufbahn beschreiben?

Maschinen im Allgemeinen

Frage: Welche Arten von Maschinen gibt es bei TRUMPF?

Frage: Gibt es eine spezielle Maschine, auf die Sie sich fokussieren?

Frage: Wer sind die Benutzer dieser Maschinen und welche Relevanz hat dies für Benutzer, Service und Inbetriebnahme?

Frage: Welches sind die unterschiedlichen Aufgaben der jeweiligen Benutzer?

Aufgaben

Frage: Was sind die täglichen Aufgaben im Laser Applications Center (LAC)?

Frage: Welche Maschinen kommen zum Einsatz und welche groben Einsatzzwecke haben diese?

Frage: Welche Herausforderungen oder Schwierigkeiten treten bei diesen Aufgaben auf?

Arbeitsprozess

Frage: Wie gestalten sich die Interaktionen generell beim Arbeitsprozess, beispielsweise mit Tools wie VisionLine oder TruControl?

Frage: Welche Arten von Daten entstehen beim Arbeitsprozess und welche Datenquellen gibt es?

Frage: Welche Informationen sind allgemein wichtig beim Arbeitsprozess XY?

Frage: Gibt es spezielle Sensoren in den Maschinen?

Frage: Existiert eine Vorbereitungsphase beim Arbeitsprozess?

Frage: Findet eine Überwachung des Arbeitsprozesses statt?

Qualität

Frage: Wie wirken sich diese Herausforderungen auf die Qualität der Schweißnaht aus und wie beeinflusst der Bediener die Qualität? Wie erkennt man dies im Arbeitsprozess?

Frage: Wie beeinflussen Vorstufen die Qualität und wie wird dies im Arbeitsprozess erkannt?

Frage: Können teilweise Arbeitsschritte vergessen werden?

Frage: Gibt es noch andere Qualitätsmerkmale?

Frage: Können Sie einen Überblick über die Informationen im Arbeitsprozess geben?

Frage: Kann die Visualisierung eine Rolle bei der Führung des Arbeitsprozesses spielen, um das Ziel zu erreichen?

Sicherheitsaspekte

Frage: Gibt es sicherheitsrelevante Informationen, die klar und deutlich visualisiert werden sollten?

Frage: Wie können Visualisierungen dazu beitragen, die Sicherheit im Umgang mit den Maschinen zu erhöhen?

Verbesserung und zukünftige Richtungen

Frage: Welche Bereiche der aktuellen Visualisierungen sehen Sie als verbesserungswürdig an?

Frage: Gibt es Technologien oder Ansätze, die in Zukunft die Visualisierung der Daten verbessern könnten?

Frage: Wie könnten zukünftige Entwicklungen in der Visualisierungstechnologie den Laserbearbeitungsprozess beeinflussen?

Frage: Sehen Sie irgendwelche Fortschritte oder Technologien voraus, die die Art und Weise der Datenerfassung oder -visualisierung im Schweißprozess verändern könnten?

Frage: Wie stellen Sie sich die Zukunft der Datenvisualisierung für den Schweißprozess mit Lasermaschinen vor?

Frage: Haben Sie eine Konkurrenzanalyse durchgeführt, um zu verstehen, wie andere Unternehmen die Datenvisualisierung für den Schweißprozess handhaben?

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Druck-Exemplaren überein.

Stuttgart, 06.12.2023

V. Brauhner

Ort, Datum, Unterschrift