Technische Universität Berlin
Fakultät Verkehrs- und Maschinensysteme
Institut für Werkzeugmaschinen und Fabrikbetrieb
Fachgebiet Industrielle Automatisierungstechnik
Prof. Dr.-Ing. Jörg Krüger

IAT: Automatisierungstechnisches Projekt WS 19/20
an der Technischen Universität Berlin

Abschlussbericht

Erweiterung eines gestengesteuerten, projektorbasierten AR-Interfaces zur benutzerfreundlichen Programmierung von Industrierobotern

vorgelegt von

Wei-Chieh Hsu (405028)
Christoph Rüger (358598)
Ulrike Schäfer (402834)
Rajmund Stankiewicz (327608)

Betreuer: Oliver Heimann

27. November 2020

Inhaltsverzeichnis

1 Einleitung

Einige kleine und mittelständische Unternehmen (KMU) mit industriellen Fertigungsprozessen haben aufgrund des demografischen Wandels und Fachkräftemangels Schwierigkeiten, ihre Fertigungsprozesse zu expandieren oder sogar aufrecht zu erhalten. Dies trifft insbesondere wegen des internationalen ökonomischen Wettbewerbes zu. An Lösungen bezüglich dieser Problematik wird bereits gearbeitet, wie z. B. Digitalisierung mittels Industrierobotern. Problematisch ist bei diesen allerdings, dass solche Lösungen gerade für KMU oft nicht praktikabel sind. Zum einen sind diese kostenspielig bzw. wirtschaftlich nicht tragbar. Zum anderen wird darüber hinaus auch Expertenwissen benötigt, um die Roboter zu programmieren. Dies stellt sich aufgrund des Fachkräftemangels derzeit oft als schwierig heraus und birgt zumeist hohe Personalkosten oder eine Abhängigkeit von externen Dienstleistern.

Genau an dieser Problemstelle setzt der Rahmen der vorliegenden Projektarbeit an: Es soll erkundet werden, inwiefern mit weniger Fachwissen eine Roboterbedienung bzw. -einstellung möglich ist, insbesondere mit Augmented Reality. Damit wäre es möglich, eine anwendungsfreundliche Roboterprogrammierung vornehmen zu können. Beispielsweise könnten so Anordnungen von Platzierungen von Schweißnähten und Bohrungen auf eine intuitive Art und Weise übermittelt werden, sodass dies auch von Laien eine derartige Programmierung umsetzbar ist. Erhofft wird sich dadurch, dass kleinere Betriebe auch bei kleinen Stückzahlen ihre Fertigungsprozesse ausbauen bzw. erhalten können.

Folglich dient dieses Projekt dem Zweck, die oben genannte Problematik effektiv zu lösen mittels: *einer methodisch entwickelten und evaluierten Sammlung von Eingabeelementen in Form einer Beispielapplikation und Vorüberlegungen für gestengesteuerte, projektorbasierte AR-Systeme in der industriellen Fertigung.*

In dieser wurde sich folglich das Ziel gesetzt, Interaktionskonzepte für die aufgabenorientierte Offline-Programmierung von Fertigungsrobotern mittels einer Projektor-basierten Augmented Reality (AR) Benutzerschnittstelle zu erarbeiten, umzusetzen und zu evaluieren. Dabei wird sich erhofft, dass Ideen und eine Beispielanwendung für einen derartigen Teilprozess entwickelt werden kann, welcher simpel und einfach zu erlernen und durchzuführen ist. Neben der Erstellung dieser, gilt es die Mensch-Maschine-Interaktion näher zu beleuchten bzw. die Usability der erstellten Anwendung zu beurteilen. Um eine gute effiziente Anwendungserfahrung zu ermöglichen, sollte darauf geachtet werden, dass die betrachteten Entwicklungsschritte mensch- und prozessorientiert durchgeführt werden können. Speziell hierfür werden im Folgenden Methoden aus der Human-Factors-Forschung bzw. des Usability Engineering angewandt. Dabei ist es nicht das Ziel

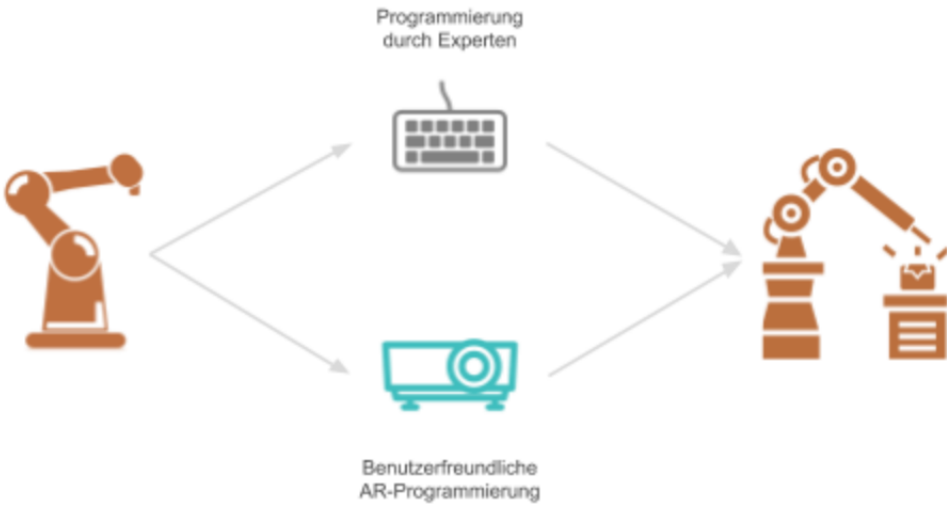

Abbildung 1.1: AR-Interfaces als Programmieralternative (Icons ©flaticons.com)

bestimmte Werte in gewissen Usabilityskalen zu erreichen, sondern die Usability auf eine sinnvolle und angebrachte Art und Weise zu erheben, um nachträgliche Probleme aufzudecken und Empfehlungen aussprechen zu können. Dafür war es nötig, verschiedene Konzepte bezüglich der Funktionalitäten und deren Darstellung zu entwickeln. Diese wurden innerhalb von Vorüberlegungen entwickelt, wobei sich im Endeffekt für eine Version zu entscheiden war, welche es nach Usability-Standards zu testen galt. Somit kann diese auf deren Praxistauglichkeit beurteilt werden.

Schlussendlich soll innerhalb dieses Projektes eine Beispielanwendung mit grundlegenden Eingabemöglichkeiten erstellt werden, welche für derartige Projektor-AR-Bedienungen im Kontext KMU-Fertigungsprozesse praktikabel sein soll. Innerhalb dieser Anwendung gilt es mehrere Funktionalitäten einzubauen, welche im Nachhinein als Gerüst für zukünftige, spezifische Anwendungen dienen können. Demnach sollen deren Inhalte eine sinnvolle Toolbox für weitere Anwendungsverbesserungen und -funktionalitäten darstellen. Folglich ist das primäre Ziel, ein robustes Anwendungsgerüst zu entwerfen anstatt einer optimierten und spezialisierten Anwendung. Die erstellte Anwendung orientiert sich an einigen Usabilitykonzepten, welche später näher beschrieben werden. Die Anwendung wurde nachträglich nochmals bezüglich dieser evaluiert, wodurch sich Empfehlungen für Folgeprojekte ergeben sollten. Explizit ausgenommen waren in diesem Projekt entsprechend Änderungen an der existierenden Anwendung und der Robotersteuerung.

2 Grundlagenrecherche zur Problemeingrenzung

Bevor die Aufgabenstellung sinnvoll eingegrenzt werden konnte, galt es eine Grundlagenrecherche durchzuführen. Hierbei wurden verschiedene Themengebiete betrachtet, um ein möglichst passendes theoretisches Fundament für das Projekt zu erarbeiten.

2.1 Ermittlung möglicher Anforderungen

2.1.1 Vorgangsparameter

Um Industrieroboter für, in der Praxis relevante, Prozesse programmieren zu können, müssen dem System bestimmte Parameter übergeben werden können. Anfangs haben wir versucht diese für das Anwendungsbeispiel des Schweißens zu bestimmen. Hilfreich waren dafür, die Arbeiten von Kuss et al., 2017, in denen ein intuitives Interface zur Programmierung von Schweißvorgängen entwickelt wurde. Dabei fanden wir heraus, dass die, für eine vollständige Beschreibung, zu übergebende Anzahl der Parameter sehr groß ist. Abhängig von der konkreten Anwendung des Systems sind Vereinfachungen und Tradeoffs üblich. Um diese zielführend einzugrenzen und auszuwählen bedarf es jedoch eines konkreten Anwendungsfalls oder Szenarios, welches uns in diesem Projekt nicht vorlag. Wir haben uns deshalb dafür entschieden, eine möglichst allgemeine und grundlegende Fragestellung zu bearbeiten und uns nicht auf konkrete Anwendungsfälle einzugrenzen. Dies würde erfordern willkürliche Annahmen zu treffen und das Arbeitsergebnis würde möglicherweise nur unter diesen gelten.

2.1.2 Vereinfachungen der Benutzerschnittstelle

Diese, bei der Vereinfachung industrieller Roboterprogrammierung auftretenden, Tradeoffs wurden von Rossano et al., 2013, systematisch untersucht. Die Forschungsgruppe hat dabei zentrale Kriterien herausgearbeitet, an welchen man sich bei der Systemgestaltung orientieren sollte:

1. Berücksichtigen der Endnutzer und den Kernbereich ihrer Programme
   - Identifizieren der Personas der Endnutzer und sorgfältige Berücksichtigung ihrer Erfahrungen mit Programmierung und Robotik
   - Identifizieren der Arten der logischen Verzweigungen, die von den Anwendun-

gen benötigt werden und sorgfältige Berücksichtigung der Entscheidungsfindungsprozesse und die Fehlerbehandlungen der Nutzer
- Identifizieren Sie die Pfadtypen, die von der Zielanwendung benötigt werden, und achten Sie dabei sorgfältig auf die Anzahl der Punkte und die Qualitätsanforderungen (z.B. die Pfadgenauigkeit).

2. Überlegen, wie die Benutzer ihre Programme bearbeiten, optimieren und debuggen. Einige Ansätze, die Code generieren, erschweren die Bearbeitung nach der Generierung.
3. Überlegen, wie der Programmieransatz die Optionen der Benutzer einschränkt und welche Optionen dem Benutzer (falls vorhanden) offen bleiben, damit fortgeschrittenere Benutzer ihre Programme außerhalb der benutzerfreundlichen Tools anpassen können. Einige Ansätze sind nur schwer erweiterbar.
4. Überlegen, wie sich die benutzerfreundlichen Tools auf fortgeschrittene Benutzer auswirken können. Einige Ansätze, die Anfängern helfen, können für fortgeschrittene Benutzer hinderlich sein.

2.2 AR User Interfaces

Zunächst gilt es, die zentralen Zieldimensionen, welche die Qualität einer Benutzerschnittstelle festlegen, für den Kontext der Roboterprogrammierung zu klären. Die Richtlinie zur Mensch-System-Interaktion (DIN EN ISO 9241) definiert hierzu Usability als den Grad, in welchem ein technisches System bzw. dessen Mensch-Maschine-Schnittstelle effektiv, effizient und zufrieden stellend verwendet werden kann. In diesen drei Zieldimensionen soll die vorliegende Arbeit zu einer Verbesserung der Bedienung und Programmierung von Robotern beitragen:

1. Effektivität bedeutet hierbei, wie genau und wie vollständig ein Benutzer die Anforderungen einer Anwendung im resultierenden Roboterprogramm abbilden kann.
2. Die Effizienz bemisst sich aus dem Aufwand, der zur Erreichung einer effektiven Programmierung notwendig ist.
3. Die Zufriedenheit des Benutzers trägt maßgeblich zur Akzeptanz des Systems bei. Darunter fallen eine ergonomische

Gestaltung, insbesondere die Vermeidung unnötiger körperlicher und mentaler Belastungen, und eine hohe Robustheit und Fehlertoleranz des Schnittstellensystems. Das Ziel des Einsatzes der AR-Technologie muss darin liegen, im Vergleich zu bekannten Systemen und Interaktionsmethoden des Standes der Technik, eine wesentliche Steigerung in Bezug auf diese drei Kriterien zu erzielen.

Eine weitere wichtige Komponente ist die Recherche bezüglich Möglichkeiten des User Interfaces, wobei neben der AR-Komponente auch allgemeinere Aspekte und Beispiele bezüglich industrieller Nutzung beleuchtet werden. Die Studie „Userinterfaces for cyber-physical systems“ von Paelke et al., 2015 beschäftigt sich beispielsweise allgemeiner mit

unterschiedlichen Assistenzsystemen und deren Interfaces in der industriellen Produktion. Für die interaktive Eingabe von komplexen geometrischen Sachverhalten, insbesondere für das Konstruieren und Modellieren sind zahlreiche Forschungsansätze in der Industrie bekannt. Die anfänglichen Ergebnisse in dieser Studie deuten darauf hin, dass der Vorteil spezifischer Interaktionstechnologien v.a. von der zu erledigenden Aufgabe selbst abhängen. Des Weiteren wird der Hinweis an nachfolgende Foschenden gegeben, dass neuzeitigere Interaktionskonzepte - wie hier AR - keinen Ersatz für herkömmliche Methoden darstellen, sondern eher als Erweiterung, um das Interface besser an die Aufgabe anpassen zu können. Folglich lässt sich für die vorliegende Untersuchung festhalten, nicht zwanghaft auf potentielle AR-Vorteile aufzubauen, wenn diese nicht der Natur der Aufgabe dienlich sind. Passung zwischen Interface und Aufgabe bzw. Aufgabenart sind hoch prioritiersiert, dabei sollte zudem versucht werden, die Vorteile von AR bestmöglich einzubinden.

2.3 Gestenbasierte Eingaben

Von Interesse sind Untersuchungen zu gestenbasierten Eingaben, da in der vorliegenden Arbeit ebenfalls eine gestenbasierte Eingabe per Zeigen mit dem Finger genutzt werden soll. Beispielsweise betrachteten Cabral, Morimoto und Zuffo, 2005 gestenbasierte Eingaben bezüglich negativer oder positiver Effekte. Ihre Resultate weisen darauf hin, dass Zeigeaufgaben via Hand langsamer sind als bisherige Methoden. Ebenso tritt bei dieser Methode vermehrt Ermüdung auf. Neben diesen negativen Aspekten stellten sich dennoch Vorteile heraus, wie der schnelle Zugang zu Umweltressourcen und einer Erhöhung der Intuition während der Nutzung. Hierbei lässt sich eine Parallele zu den in Kapitel erwähnten AR-Interfaces ziehen. Sowohl bzgl. AR-Interfaces als auch gestenbasierter Eingabe, lassen sich Vor- und Nachteile erkennen, wobei je nach Aufgabe abzuwägen ist, ob die Vorteile die Nachteile überwiegen und inwiefern die Methoden genutzt werden sollen. Neben der eben genannten Studie beschäftigten sich weitere Forscher u.a. mit dem Vergleich zwischen Arm- und Fingerzeichen für Desktopapplikationen (Farhadi-Niaki, Etemad und Arya, 2013). Es stellte sich heraus, dass Fingerzeichen dabei besser als Armzeichen sind. Diese Erkenntnis unterstützt die vorliegende Arbeit bzgl. der Auswahl des einfachen Zeigens mittels Fingers, da mittels Fingerzeichen Präzision, Effizienz und Ease-Of-Use angemessen zu sein schienen. Selbstverständlich fokussierte diese Studie eine andere Grundthematik, dennoch deuten die Ergebnisse auf eine prinzipielle Nutzbarkeit der gestengesteuerten Eingabe mittels Finger hin. Zudem kann an dieser Stelle bereits bzgl. der Usabilitytestung festgehalten werden, dass in dieser Studie ebenfalls eine kleine Stichprobe (10 Personen) gewählt wurde. Dies unterstützt die später näher beschriebene Annahme, dass auch mittels kleinere Stichproben effizient Problemstellen herausgestellt werden können.

2.4 Theoretische Aspekte der Usability

Bezüglich der Usability des Interfacedesigns wird sich u.a. auf allgemeine Richtlinien siehe Mazumder und Das, 2014 bezogen. Spezifisch werden im Folgenden zum einen Shneidermans 8 Goldene Regeln erläutert (Dix et al., 2003). Zum anderen werden die Gestaltprinzipien bzgl. visueller Wahrnehmung betrachtet, welche sich v.a. mit der optischen Gestaltung des Interfaces befassen (Mazumder und Das, 2014).

An dieser Stelle werden die 8 Shneidermanschen Regeln aufgelistet und wenn nötig kurz zusammengefasst:

1. Konsistenz - Konsistente Gestaltung von Eingaben, Layout usw. zur Erhöhung der Usability
2. Shortcuts - Näherbringen von Shortcuts, Macros etc. an häufige Nutzer für mehr Effizienz
3. Informatives Feedback
4. Abgeschlossenheit - Gewährleisten des Überblicks über aktuelle Aktivität wie Abgeschlossenheit einer Aufgabe, Bearbeiten eine Aufgabe usw. für den Nutzer
5. Fehler vermeiden - Systemgestaltung soll grobe Fehler ausschließen und auftretende Fehler mitteilen/revidieren
6. Umkehrbarkeit - Gemachte Eingaben rückgängig machen
7. Benutzerkontrolle gewährleisten
8. Kurzzeitgedächtnis entlasten

Shneidermans 8 Goldene Regeln können als Sammlung usabilityrelevanter Aspekte gesehen werden. Neben derartigen eher funktionellen Betrachtungen sind Gestaltprinzipien der visuellen Wahrnehmung von Interesse. Mazumder und Das, 2014 listen diese wie folgt auf:

1. Gesetz der Nähe
   Räumlich nahe Objekte werden als zusammengehörig wahrgenommen
2. Gesetz der Ähnlichkeit
   Objekte die ähnlich aussehen, erscheinen zusammengehörig
3. Gesetz der guten Fortsetzung
   Objekte, die auf einer kontinuierlichen Linie o.ä. angeordnet sind, werden als zusammengehörig wahrgenommen
4. Gesetz der Geschlossenheit
   auch wenn Informationen fehlen (wie Lücken in einem umgebenden Rechteck), kann dies als Gesamtfigur wahrgenommen werden
5. Gesetz der Einfachheit
   auch kleine Objekte können als Figur gegenüber des Hintergrundes erscheinen
6. Gesetz der Symmetrie

Symmetrische Figuren agieren als Figuren gegenüber einem asymmetrischen Hintergund

7. Gesetz der Umgebenheit
   Objekte, welche von anderen umschlossen werden, können als eine Figur angesehen werden
8. Gesetz der Prägnanz
   Im visuellen Feld werden immer einige Objekte prominenter und andere eher als Hintergrundrollen wahrgenommen

Weiterhin, laut Hornbogen, 2018, hilft es, zur Usabilityverbesserung auch aufgestellte Dialogprinzipien laut DIN EN ISO 9241-110 bezüglich des Dialogs mit einem interaktiven Design zu beachten. Es handelt sich dabei der Informationsautausch zwischen dem Nutzer und dem Softwaresystem, wobei die Dialoggestaltung die ergonomiesche Systementwicklung und gleichfalls die Bewertung von Benutzerschnittstellen (zwischen user und Interface) umfasst. Diese stammt aus dem Bereich der Normung von Benutzungsschnittstellen zur Verbesserung von Gebrauchstauglichkeit. Insgesamt werden sieben Prinzipien beschrieben:

1. Aufgabenangemessenheit
   - passende Unterstützung des Dialogsystems für den Nutzer zur expliziten Arbeitsaufgabenerledigung
2. Selbstbeschreibungsfähigkeit
   - Software zeigt dem Nutzer an, wo sich dieser zuzeit im System befindet und welche Hanldungen wann und in welcher Art ausführbar sind
   - Umgang mit komplexer Systemarchtitetktur durch intuitives Dialogsystem erleichtern
3. Erwartungskonformität
   - vom Nutzer ausgelöste Aktionen müssen diesem klar und unmittelbar zur Nutzeraktion angezeigt werden zum jeweiligen Aufgabenkontext
   - ermöglichen von Einschätzung von Auswirkung und Dauer einer Aktion
4. Lernförderlichkeit
   - lernförderliche Gesaltung zum korrekten Bedienen der Software
   - Hilfsfunktionen als Unterstützung
5. Steuerbarkeit
   - zum Zielerreichen muss dem Nutzer im Dialog mit dem System möglich sein, eine Aktion einleiten, lenken und beeinflussen zu können
6. Fehlertoleranz
   - tolerantes System gegenüber Fehlern des Nutzers
   - Fehler im Voraus vermeiden / nicht zulassen
   - einfache Korrekturen ermöglichen
7. Individualisierbarkeit
   - individualisierbares System z.B. im Sinne der Barrierefreiheit

Neben diesen können die Prinzipien des Informationsdesigns aus ISO 9241-12 helfen nach Akkreditierungsstelle, 2010, handlungsleitende Informationen passend darzustellen und zu designen und zusätzlich die Aufmerksamkeit des Nutzers angemessen zu lenken. Hierbei handelt es sich um drei Kriterien:

1. Erkennbarkeit
   Nutzeraufmerksamkeit auf gewünschten Information lenken
2. Unterscheidbarkeit
   Information von Interesse sollte einfach von anderen Daten unterscheidbar sein
3. Lesbarkeit
   Leicht lesbare und verständliche Information

Eine weitere mögliche Hilfe sind validierte Usability-Verfahren wie die Usability-Scale SUS (Brooke et al., 1996). Diese ist jedoch ein sehr allgemeines Instrument zum Erfassen der Usability, wodurch es schwierig ist, genaue Verbesserungsideen u.s.w. mittels dieser herausstellen zu können. Es werden innerhalb dieser beispielsweise derartige Aussagen abgefragt: “Ich empfinde die Bedienung als sehr umständlich”. Somit wäre es möglich generelle Problemstellen aufzudecken.

Letztendlich sind all diese Prinzipien eher generelle Richtlinien, die dabei behilflich sein können Anwendungen zu planen und zu testen. Dies impliziert ebenfalls, dass nicht alle genannten Konzepte umzusetzen sind. Je nach Kontext gilt es zu entscheiden, welche zu beachten sind.

3 Vorüberlegungen

Im Folgenden wird sich beispielhaft auf Schweißnähte oder auch Bohrungen bezogen. Diese waren die zuerst angedachten Anwendungsgebiete und standen demnach im Zentrum der Vorüberlegungen. Jedoch wechselte der Fokus des Projekts letztendlich auf die Eingabe von Geometrieelementen. Dies geschah aus zwei Gründen: Einerseits waren die konkreten Anwendungsfälle noch nicht festgelegt; andererseits sind für quasi alle Fertingsprozesse, bei denen der Roboter eingesetzt werden können auf Geometrieeingaben angewiesen. Segmente werden somit teilweise als Schweißnähte bezeichnet: hierbei handelt es sich aber lediglich um spezifisch ausgewählte Segmente bzw. Abschnitte entlang einer Bauteilkante. Zudem wird die Platzierung von Punkten betrachtet, welche beispielsweise bei Bohrungen erforderlich ist. In Folgenden Kapitel 5 wird hingegen nur noch von den eigentlichen geometrischen Eingaben, also Segmenten und Punkten geschrieben, da diese das eigentliche Produkt diese Projektes sind.

3.1 Vorliegende Segmentauswahl

Bevor Konzepte bezüglich des zu erstellenden Prototypen entwickelt werden konnten, war es notwendig die bisherige Funktionsweise der bestehenden Software zu verdeutlichen: So ist es möglich im ausgewählten Menü ein Segment mittels Fingerzeigen auszuwählen. Dafür werden auswählbare Kanten bereits im AR-Menü visualisiert. Der Nutzende kann daraufhin einen Punkt mittels längerer Fokussierung auswählen und von diesem aus ziehend das zu definierende Segment definieren. Wenn nun gewünscht wird, ein zweites Segment anzugeben, kann dies mithilfe erneuter Startpunktauswahl und anschließenden Ziehen des Fingers entlang der auszuwählenden Kante geschehen. In wird diese Segmentsetzung exemplarisch dargestellt. Die Anfangs- und Endpunkte existieren dabei nur zur Veranschaulichung in unserer Grafik und werden nicht in der Software festgelegt. Es ergibt sich ein ausgewähltes Segment als Ergebnis wie in der folgenden Abbildung zu sehen ist:

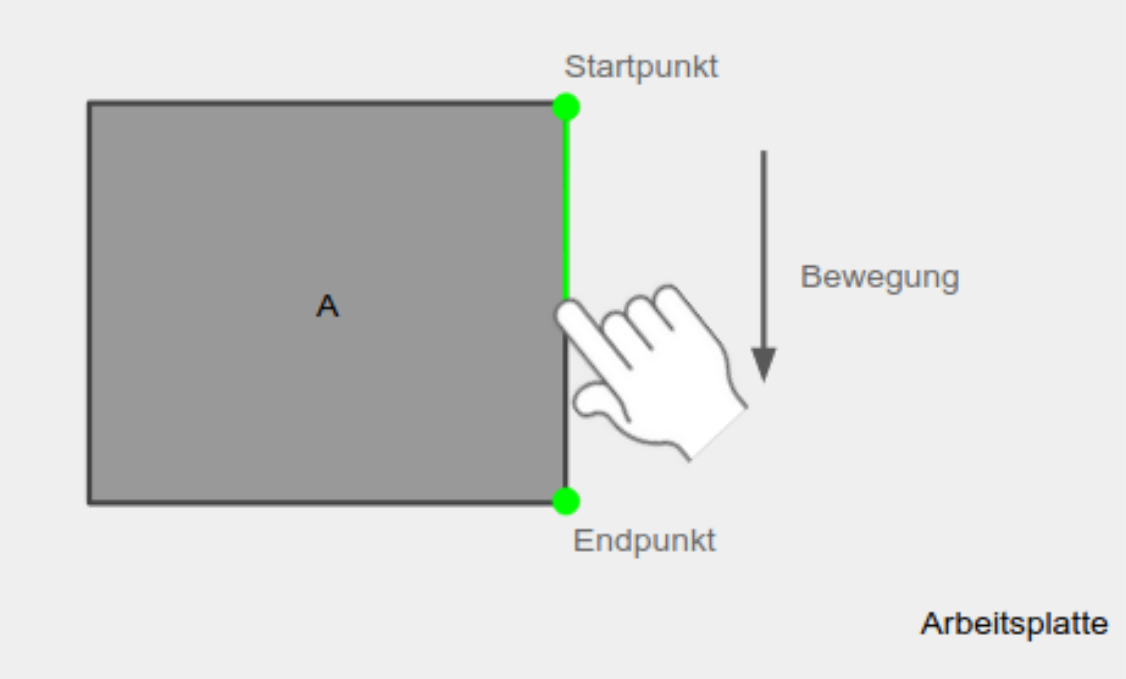

Abbildung 3.1: Darstellung des bestehenden Ansatzes der Segmentauswahl

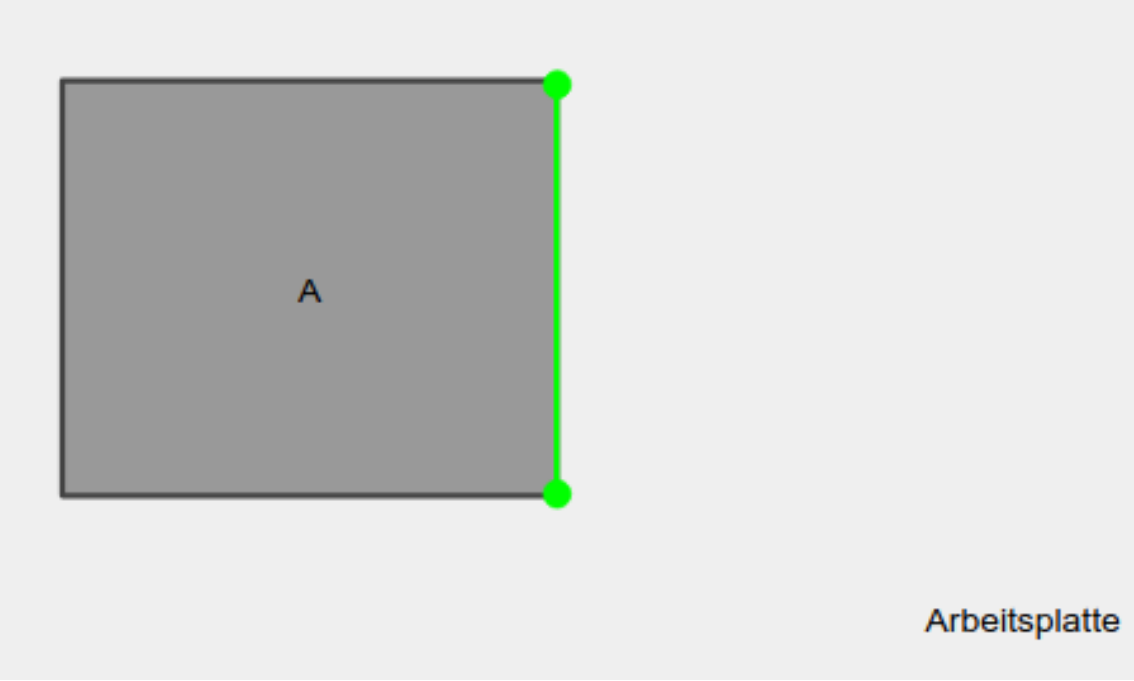

Abbildung 3.2: Ergebnis der herkömmlichen Segmentauswahl

3.2 Abgeleitete, spezifische Problemstellung der Segmentauswahl

Ein Problem bei dieser Methode ist, dass sich Anfangs- und Endpunkte der Schweßsegmente nicht exakt definieren lassen. Das wohl größte Hindernis dabei ist die Fingerbreite des Nutzenden, welche quasi der maximalen Ungenauigkeit der Punktplatzierung ent-

spricht. Zudem ist es momentan nicht möglich, bereits getätigte Eingaben im Nachhinein zu korrigieren. Sie können lediglich gelöscht und erneut eingegeben werden. Ebenfalls nicht möglich wäre es beispielsweise Segmente, welche derzeit keine ausgewählten z.B. Schweißsegmente sind, als Schweißsegmente umzudeklarieren oder umgekehrt.

Zusammengefasst ergibt sich die Problemstellung:

Wie kann man spezifisch auswählen, welche Kantensegmente ausgewählt werden sollen und welche nicht? Und wie können die Segmente exakt definiert werden?

3.3 Konzeption der Segmentauswahl im Prototypen

In diesem Abschnitt gilt es prinzipielle Ideen bezüglich der Segmentauswahl im Prototypen strukturiert festzuhalten. Grundstein des neuen Ansatzes ist es, nicht mehr von direkter Segmentsetzung auszugehen; das neue Grundkonzept basiert stattdessen auf Setzung von Start- und Endpunkten. Zudem soll auch eine Möglichkeit geschaffen werden, um Punkte auf dem Werkstück zu definieren.

3.3.1 Streckenauswahl - Punktmenü

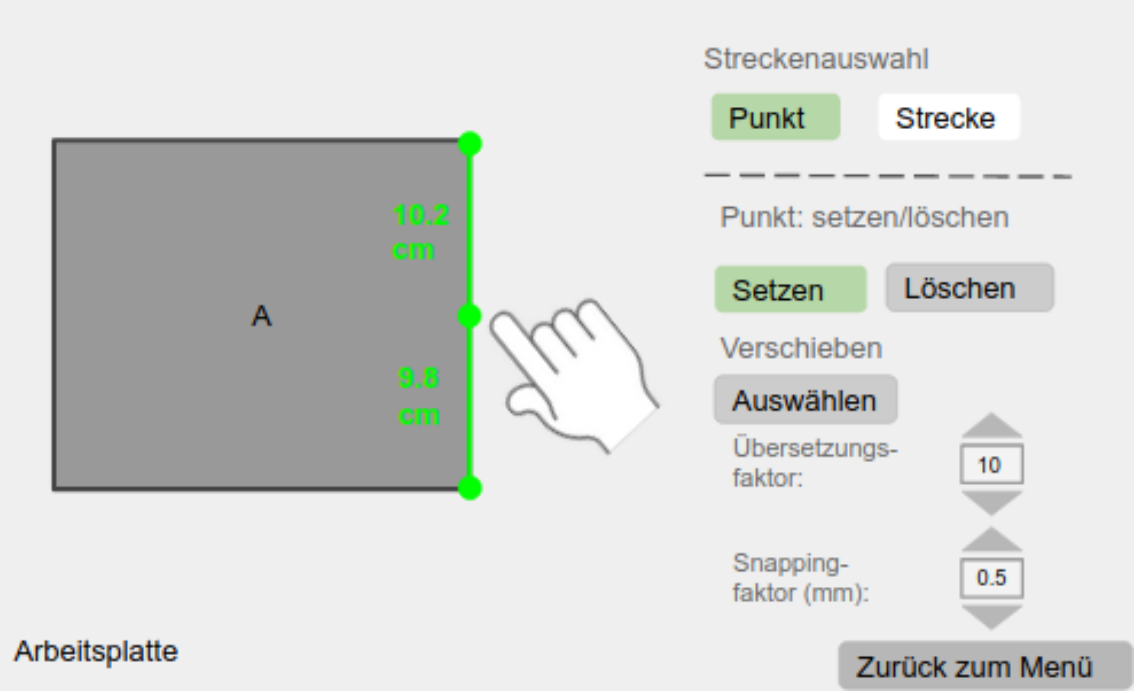

Abbildung 3.3: Punktmenü - Punktsetzung

Wie in der obigen Abbildung zu sehen ist, ist geplant, dass im Streckenmenü generell zwei Buttons angegeben werden. Zum einen der Button “Punkt”, welcher zum Punktmenü führt und zum anderen der Button “Strecke”, welcher das Streckenmenü erscheinen lässt. In Abbildung 3.3 wird in diesem Fall das Punktmenü angezeigt. Grün markierte Buttons sollen hierbei den Status “ausgewählt” in der Konzeptdarstellen darstel-

len. Prinzipiell wäre es zudem gut, wenn es einen “Zurück zum Menü”-Button gibt, um zum Hauptmenü zurück zu gelangen. Da die geplante Anwendung vorerst unabhängig vom Hauptprogramm konstruiert wird, wird im folgenden nicht auf weitere Aspekte der Hauptmenüs und der Verbindung zu diesem eingegangen.

Das Punktmenü untergliedert sich anfangs in die Funktionen “Punkt: setzen/löschen” und “Verschieben”. Wenn die Funktion “Setzen” aktiviert ist, bedeutet dies, dass mit längerem Fingerzeigen es nun möglich ist auf eine vordefinierte Kante / Ecke einen Punkt zu setzen. Dabei sollen zudem Maße angezeigt werden, welche helfen,die richtige Position auszuwählen. Selbstverständlich muss der Setzfunktion als Pendant eine Löschfunktion gestellt werden. Wenn diese ausgewählt und der betreffende Punkt fokussiert wird, wird dieser anschließend gelöscht. Neben dem bloßen Erzeugen und Löschen, soll es möglich sein, bereits gesetzte Punkte zu verschieben. Dafür muss der Button “Auswählen” aktiviert und ein Punkt ausgewählt werden. Wenn man nun den Finger entlang der Kante, auf welcher der Punkt gesetzt wurde, bewegt, wird der Punkt ebenfalls in diese Richtung verschoben. Wie vorhin erwähnt, ist es schwierig, genaue Position zu definieren aufgrund der Breite des Fingers und potentiell ungenauer Bewegungen. Für die Problematik wurden unter dem Button “Auswählen” die beiden Funktionen “Übersetzungsfakto” und “Snappingfaktor” ergänzt. Erstere dient v.a. der Aufgabe, geoße Fingerbewegungen in kleine Punktverschiebungen zu übersetzen. Der Angegebene Übersetzungsfaktor von 10 sollte beispielsweise bei einer tatsächlichen Fingerbewegung von 10cm eine Verschiebung des Punktes von lediglich einem Centimeter erzeugen. Somit sollte es möglich sein, trotz gröberer Bewegungen spezifische Positionierungen zu einstellen zu können. Noch nicht definiert wurde dabei, in welchen Schritten sich der Punkt dabei verschieben kann. Mittels Snappingfaktor kann dies festgelegt werden. In Abbildung 3.3 wurde exemplarisch der Wert von 0.5mm angegeben. Dies bedeutet, dass der Punkt sich immer in 0,5mm-Schritten entlang der Fingerbewegung bewegt. Dadurch wird es einfacher, zwischen genauen und groben Verschiebungen zu springen und genaue Werte zu erreichen.

An dieser Stelle muss nochmals erwähnt werden, dass es sich hier um prinizipielle Konzepte handelt und demnach der erstellte Prototyp von diesen abschweichen kann. Hiermit sollen lediglich geplante Funktionen veranschaulicht werden.

3.3.2 Streckenauswahl - Streckenmenü

Das zweite auszuwählende Menü wird in Abbildung 3.4 dargestellt. In diesem ist es möglich Segmente setzen und löschen zu können. Beispielhaft wird dabei eine Objekt mit bereits vier festgelegten Punkten gezeigt. In diesem Beispiel sollen bisher alle drei Segmente - die bereits grün markierten - und das Segment, auf welches der Finger zeigt, ausgewählte Segmente (z.B. Shcweißsegmente) sein. Ziel in diesem Exempel ist es, das mittlere Segment als Nicht-Segment umzudefinieren. Dafür muss der Button “Löschen” mittels Fingerzeigen ausgewählt werden. Da dieser in Abbildung 3.4 bereits grün markiert ist, bedeutet dies, dass dieser aktuell bereits ausgewählt wurde. Die dargestell-

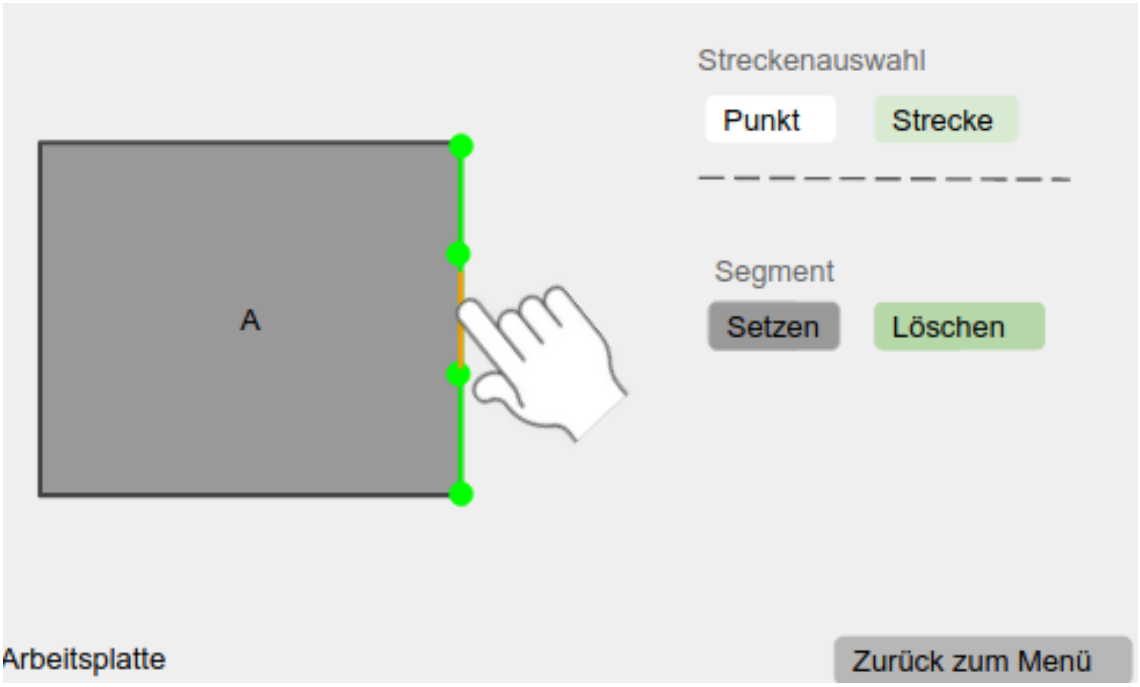

Abbildung 3.4: Streckenmenü - Streckenlöschung

te Hand muss nun das mittlere Segment fokussieren, woraufhin dieses als z.B. Nicht-Schweißsegment deklariert wird. Das gewünschte Ergebnis wird in Abbildung 3.5 gezeigt. Selbstverständlich können mittels "Setzen" beispielsweise Nicht-Schweißsegmente zu Schweißsegmenten umdeklariert werden.

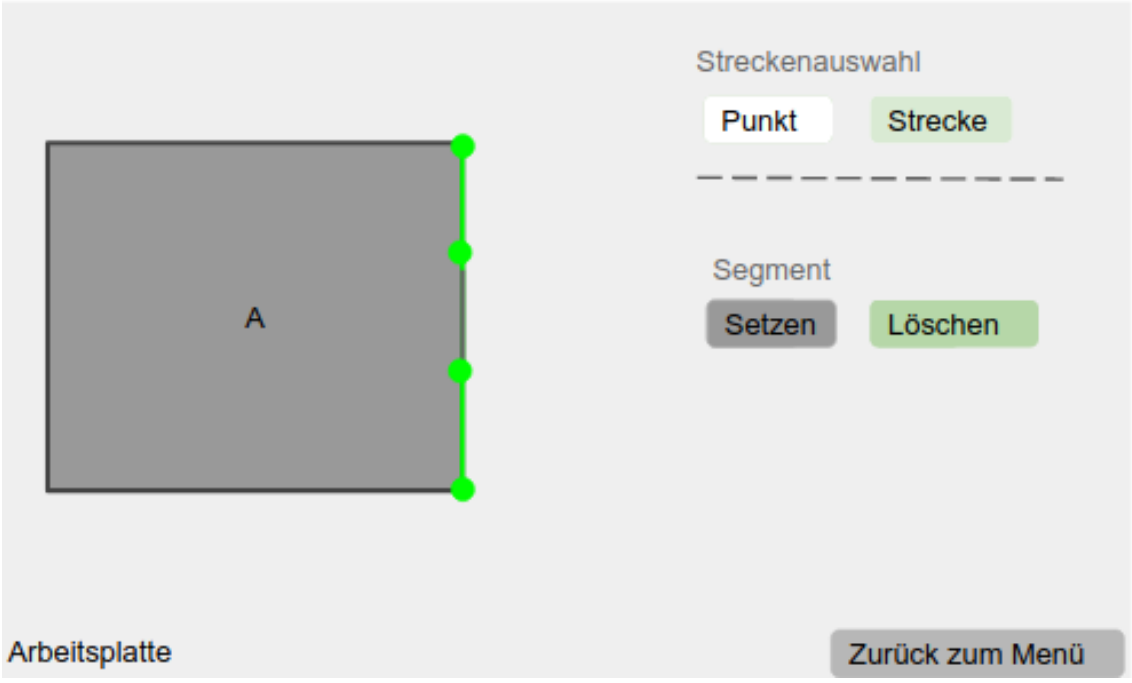

Abbildung 3.5: Streckenmenü - Streckenmenübeispiel

3.3.3 Streckenauswahl - Usabilityüberlegungen

Zudem wurden sich vor der Prototyperstellung Gedanken über die Gestaltung der Elemente, Symbole und Menüangaben gemacht. Ein Beispiel hierfür befindet sich in Abbildung 3.6

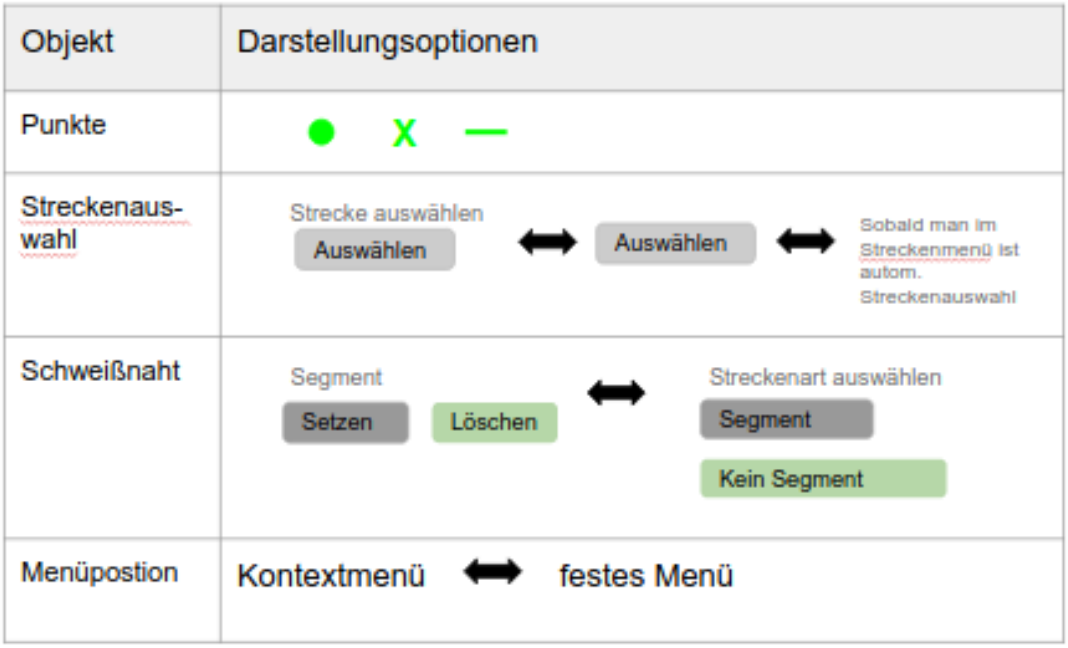

Abbildung 3.6: Usabilityüberlegungen

Entschieden werden kann sich dabei z.B. wie die Punkte dargestellt werden könnten. Ebenfalls kann das Streckenauswahlmenü auf verschiedene Arten und Weisen angezeigt werden. Dabei könnte zum einen der Untertitel “Strecke auswählen” weggelassen werden, da der Buttonname bereits intuitive ist. Auch könnte es eingestellt werden, dass per Menüauswahl bereits die Funktion “Auswählen” aktiviert ist. Ebenso gilt es Entscheidungen über Buttonnamen zu treffen. Die Segment- bzw. Schweißnahtbestimmung könnte dabei die Buttons “Setzen” und “Löschen” oder “kein Segment” und “Segment” haben. Es gilt hierbei Konsistenz zu bewahren. Das heißt, eine Bezeichnung für eine spezifische Funktion im Punktmenü sollte wenn möglich im Streckenmenü genauso betitelt sein. Von daher wird an diesem Punkt eher die erste Option bevorzugt. Eine weitere Entscheidung kann über die grundsätzliche Darstellung des Menüs getroffen werden. Dieses könnte zum einen ein festes Menü z.B. - wie bisher dargestellt - am rechten Rand haben. Alternativ könnte allerdings auch ein flexibles Kontextmenü genutzt werden. Wenn dabei beispielsweise ein Punkt fokussiert wird, könnte sich dieses Kontextmenü öffnen mit passenden Funktionen wie Löschen und Verschieben.

Insgesamt gilt es, ein prototypisches Beispiel im Rahmen des Projektes zu erstellen, weshalb sich im Laufe der Programmierung auf bestimmte Varianten geeinigt wurde, welche als besonders passend für das Projekt und für eine anschließende Usabilitytestung von der Projektgruppe erachtet wurden.

3.4 Überlegungen bezüglich des zu nutzenden Koordinatensystems

Benutzer sollen Punkte in 3D in der AR-Anwendung eingeben können. Hierbei gibt es viele mögliche Konzepte zur Umsetzen, wobei das für den Punkt benötigte Koordiantensysteme wichtig ist. Hierfür werden 4 Möglichleiten vorgestellt.

3.4.1 Punktsetzung - Existierendes Koordinatensystem - Arbeitsplatte

Grundlage dieses Ansatzes ist ein Koordinatensystem, das fest zur Arbeitsplatte ist, z.B. ein fest montierter Anschlag. Der einzugebende Punkt wird bei dieser Lösung demnach relativ zu diesem Punkt bestimmt. Dies ist programmatisch relativ einfach, erfordert jedoch, dass die Bauteile immer gleich und exakt relativ zum "Plattenursprung" positioniert werden.

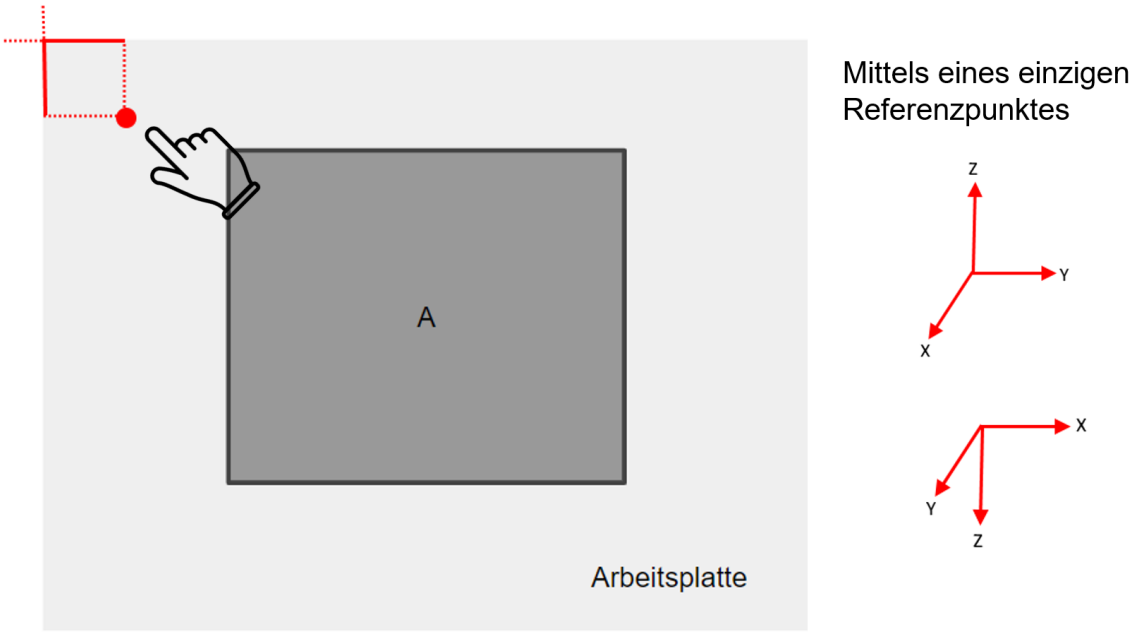

Abbildung 3.7: Existierendes Koordinatensystem - Arbeitsplatte

3.4.2 Punktsetzung - Existierendes Koordinatensystem - Bauteil

Alternativ könnte ein dem Bauteil inhärenter Ursprung definiert werden. Dieser müsste z.B. beim erstmaligen Bearbeiten des Bauteils festgelegt werden - möglicherweise über eine der beiden folgenden Methoden.

3.4.3 Punktsetzung - Selbst definiertes Koordinatensystem

Im Gegensatz zu den obigen Ansätzen müssen Nutzer vor der Punkteingabe zuvor ein Koordinatensystem definieren bzw. auswählen.

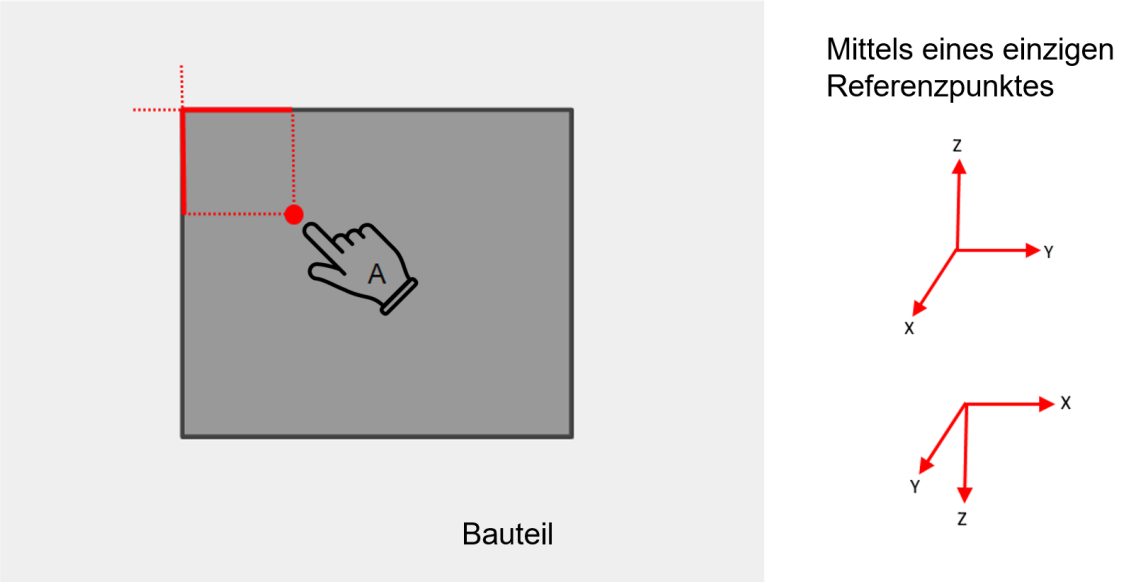

Abbildung 3.8: Existierendes Koordinatensystem - Bauteil

3.4.3.1 Punktsetzung - Selbst definiertes Koordinatensystem - Bauteilkante

Nachdem der Benutzer eine Kante des Bauteils auswählt, kann der Punkt durch einer Position entlang der Kante und einem Abstand zur Kante definiert werden.

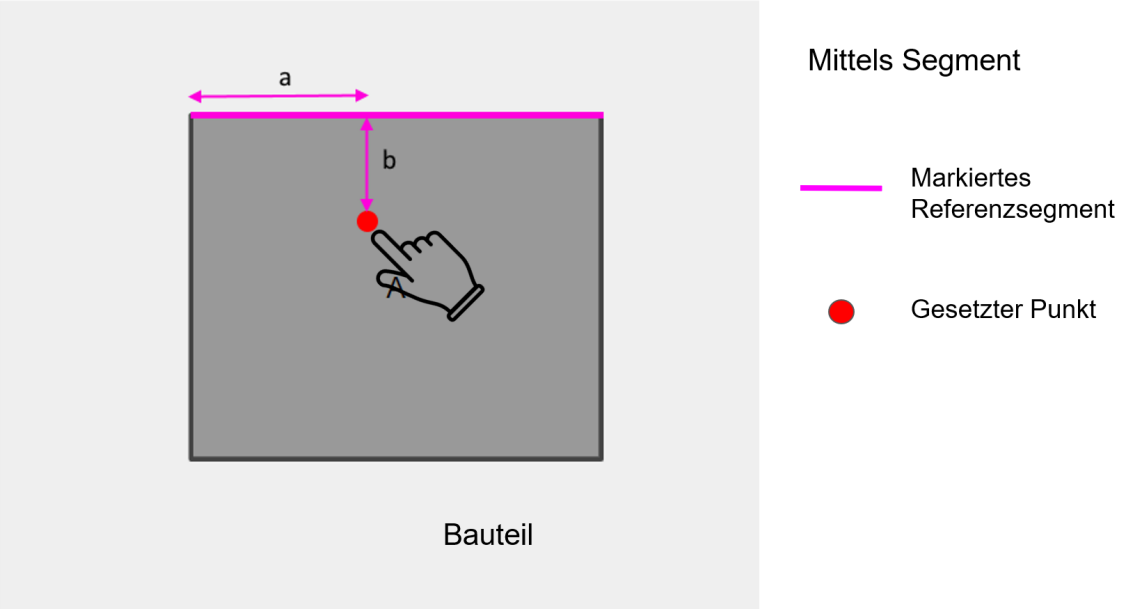

Abbildung 3.9: Selbst definiertes Koordinatensystem anhand einer Bauteilkante

3.4.3.2 Punktsetzung - Selbst definiertes Koordinatensystem - Drei-Punkte-Koordinatensystem

Indem die Benutzer drei Referenzpunkte relativ zur Arbeitsplatte bzw. den Bauteil eingeben, können entsprechend Achsen und darüber ein Koordinatensystem festgelegt wer-

den (Siehe3.10).

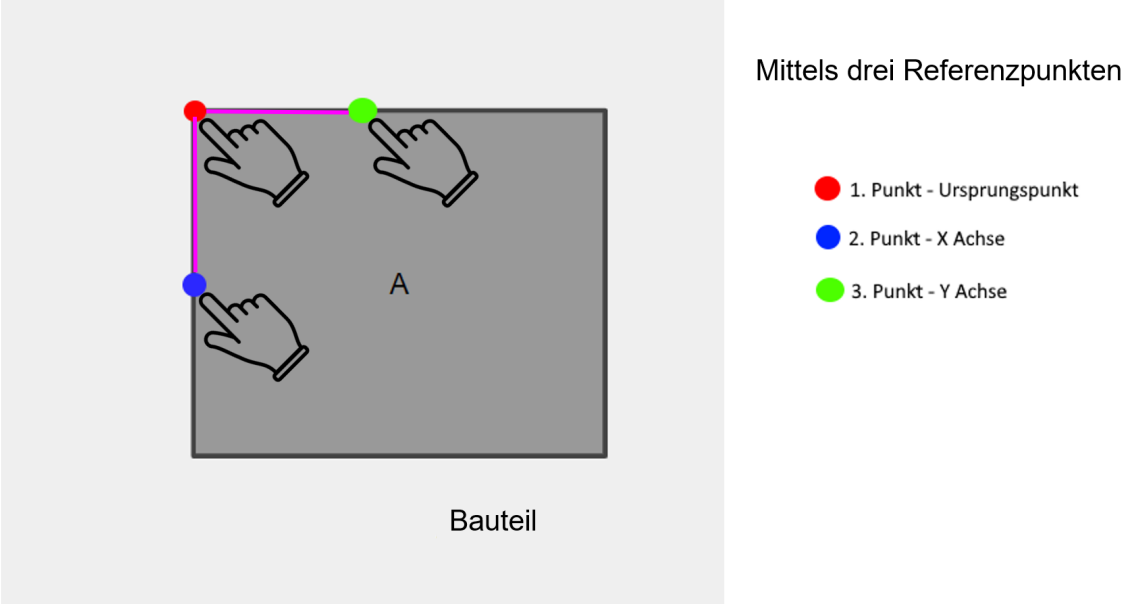

Abbildung 3.10: Selbst definiertes Koordinatensystem anhand dreier Punkte

4 Anwendungsbeschreibung

Auf Grundlage der Vorüberlegungen wurde eine Anwendung zur Erstellung und Änderungen von Punkten und Kantensegmenten erstellt. Bevor auf den Code eingegangen wird, gilt es zuerst die erstellte Anwendung (aus der Benutzerperspektive) oberflächlich vorzustellen. Orientiert wird sich hierbei an Kapitel 3. Dies fasst dabei allerdings die ersten Überlegungen zusammen, demnach gilt es nicht, diese Vorüberlegungen im Detail eins zu eins zu übernehmen, sondern die nutzbarsten Elemente und Strukturen zu beachten und einzubinden. Im Folgenden werden sowohl das Hauptmenü der erstellten Applikation als auch deren Untermenüs - Punkt- und Abschnitts- bzw. Streckenmenü - vorgestellt.

4.1 Hauptmenü

Das Hauptmenü ist der Ausgangspunkt der Anwendung und dient zur Übersicht der möglichen Aktionen, sowie der bereits erstellten Geometrien. Das im Projekt erstellte Hauptmenü der Punkt- und Streckensetzung mit gewünschten Spezifikationen ist in Abbildung 4.1 zu sehen. Das Menü - sowohl Haupt- als auch Untermenüpunkte - befinden sich auf der rechten Seite der Benutzeroberfläche und würden demnach in der AR-Projektor-Anwendung rechts neben dem Bauteil abgebildet werden. Im Hauptmenü können die Untermenüpunkte ausgewählt werden, bei welchen zum einen das Erstellen und zum anderen das Ändern von entweder Punkten oder Abschnitten ermöglicht wird. Neben diesen Funktionalitäten befindet sich oberhalb des Bauteiles die Bezeichnung des aktuellen Menüs, um dem Anwender einen besseren Überblick zu gewährleisten. Zudem werden direkt unter dieser Betitelung Anweisungen bzw. Tipps angezeigt, welche dem Anwender helfen sollen, die Anwendung korrekt nutzen zu können.

4.2 Punktmenü

Im Punktmenü können sowohl neue Punkte erstellt als auch existierende Punkte geändert werden. Zum Erstellen eines Punktes wählt man zuerst die Kante aus, welche als Referenzsystem dient.

Nach Festlegen der gewünschten Position, welche durch die Abstandsangaben in Millimetern unterstützt werden, kann der gesetzte Punkt bestätigt werden mit dem Button “Bestätigen”. Wenn ein Punkt bestätigt wird, wird der Anwender in das Hauptmenü

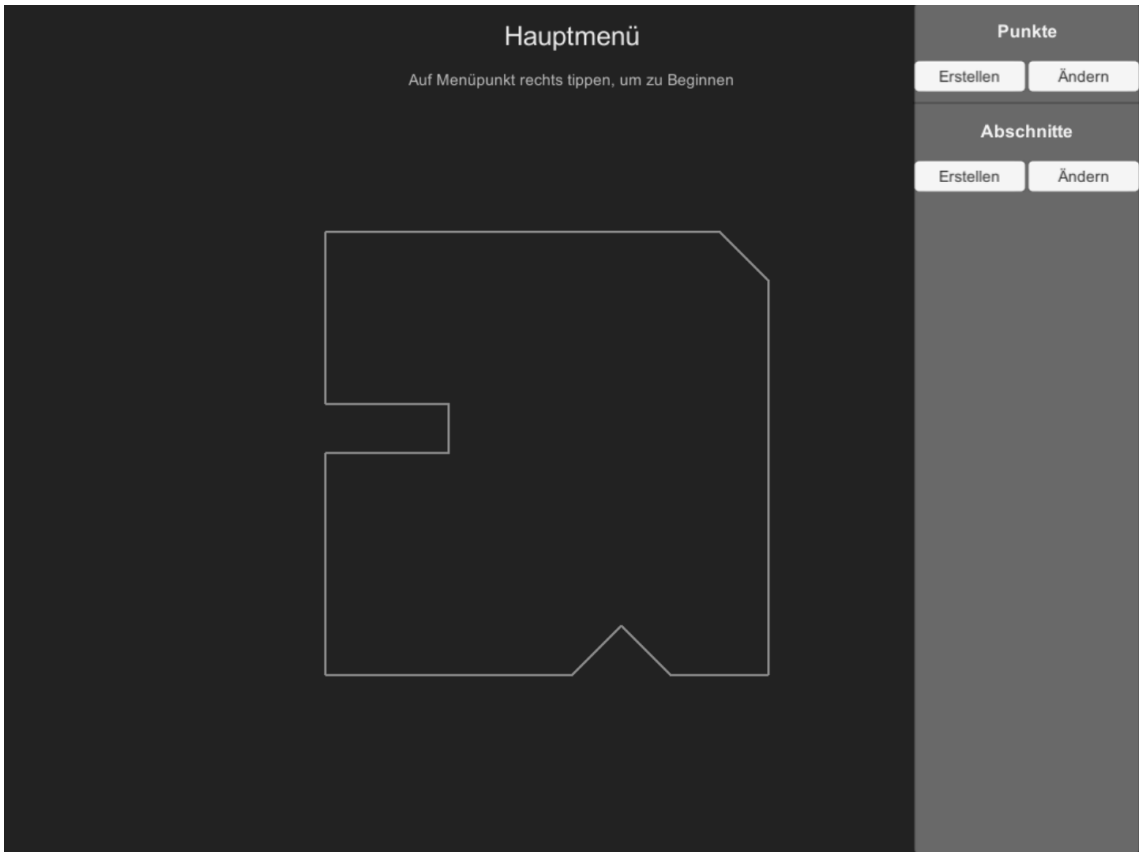

Abbildung 4.1: Hauptmenü

zurück geleitet mit dem neu erstellten Punkt. Wie in Abbildung 4.1 zu sehen, kann neben dem "Erstellen" des Punktes im Hauptmenü auch auf den Button "Ändern" geklickt werden. Dabei ist es möglich einzelne Punkte von Interesse zu markieren und diese anschließend zu verschieben, um Verbesserungen oder Änderungen vorzunehmen, wobei selbstverständlich wieder die Einrast- und Übersetzungsfunktionen genutzt werden können. Ebenso ist es möglich bereits erstellte Punkte mit "Löschen" wieder zu entfernen.

4.3 Streckenmenü

Wie im Hauptmenü (siehe 4.1) sichtbar ist, können neben Punkten auch Abschnitte bzw. Segmente gesetzt werden. Dafür muss im Teilmenü *Abschnitte* auf den Button *Erstellen* gedrückt werden. Der Anwender kann nun die Strecke erstellen - durch das Setzen eines Anfangs- und Endpunkts. Nachdem diese erstmalig gesetzt wurden, wird direkt in den Kanten-Änderungsmodus gewechselt. Hierfür stehen nicht nur (wie bei der Punktbearbeitung) die Funktionen *Übersetzungsfaktor* und *Einrasten* zur Verfügung, sondern auch *Richtung umkehren* und *Abschnitt invertieren*, s. 4.4. *Richtung umkehren* invertiert die (zuerst automatisch festgelegte) Richtung der Strecke um; *Abschnitt invertieren* bewirkt, dass Segmente zu Nicht-Segmenten und Nicht-Segmente zu Segmenten werden. Die Bestätigung der Eingabe erfolgt wie auch im Punktmenü. Ebenso wie im Punktmenü, kann im Hauptmenü statt *Erstellen* auch *Ändern* gewählt werden, wodurch es möglich ist, Stre-

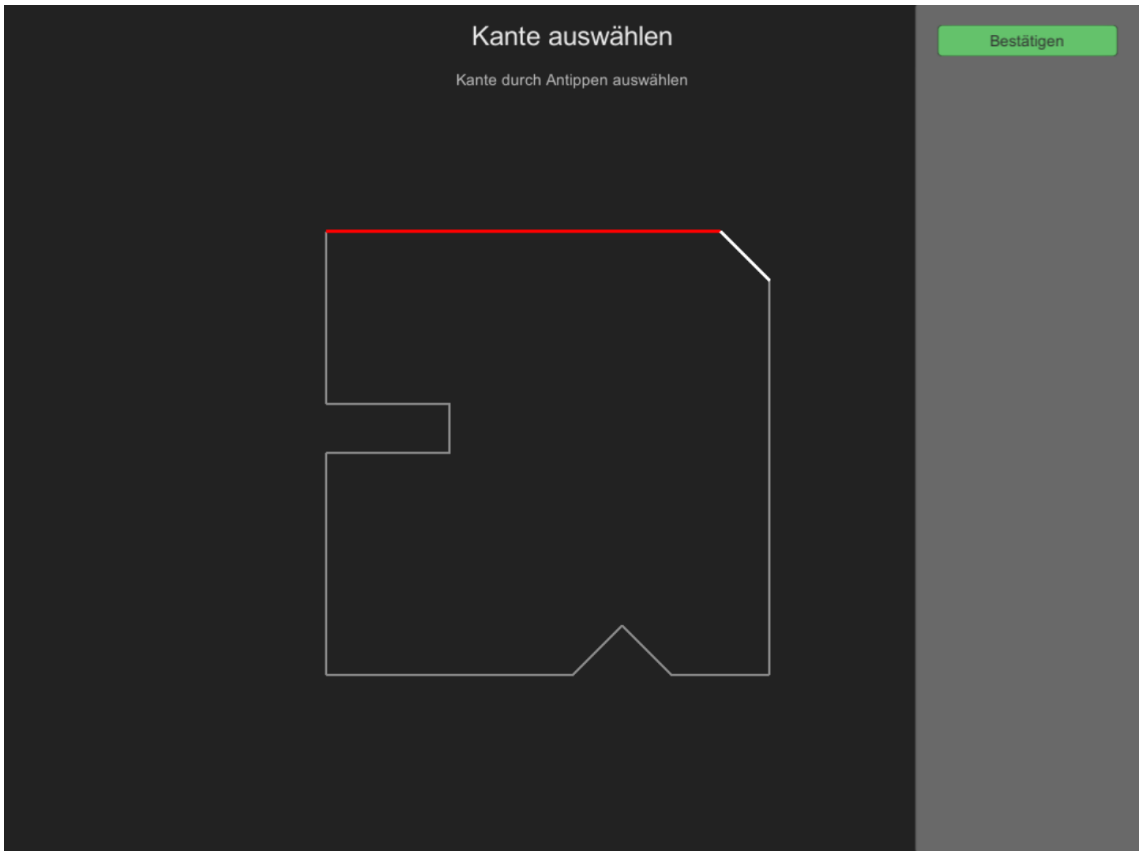

Abbildung 4.2: Punktsetzung - Kantenauswahl

cken beispielsweise zu kürzen. Selbstverständlich werden auch hierbei Abstandsangaben in Millimetern angezeigt, damit der Nutzer das Segment korrekt festlegen kann.

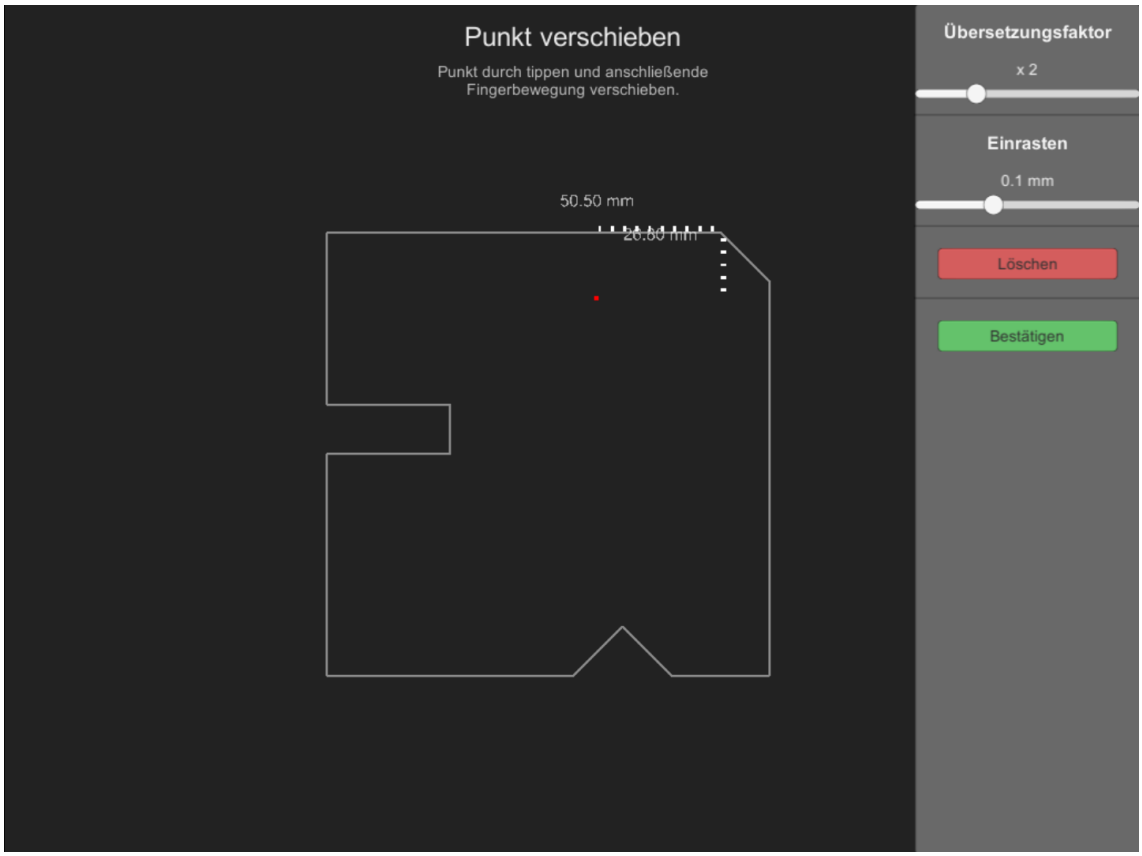

Abbildung 4.3: Punktsetzung - Punktverschieben und -erstellen

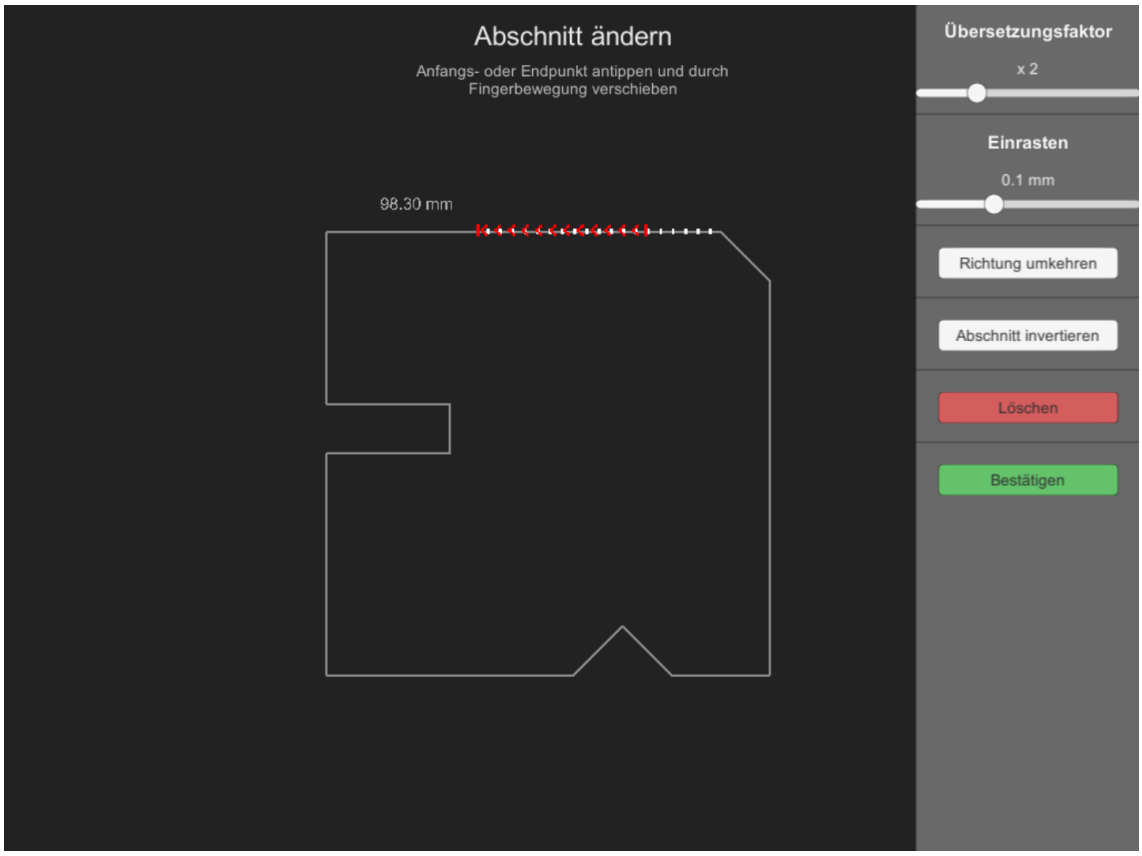

Abbildung 4.4: Streckensetzung - Abschnitteinsetzen

5 Beschreibung des Codes

Die initiale Idee für dieses Projekt ware eine Erweiterung der Benutzeroberfläche für einen existierenden Prototypen. Dieser Prototyp wurde mit Unity erstellt und hatte als exemplarischen Anwendungsfall das Setzen von Schweißnähten verfolgt. Wie in **??** dargelegt, wählten wir statt einem konkreten Anwendungsfall eine Verbesserung der eher abstrakten Geometrieeingabe. Zusätzlich war nicht sicher, ob sich andere Schnittstellen (z. B. die der Robotersteuerung) im Laufe des Projektes ändern, da möglicherweise andere Teams an diesen arbeiten. Nach Sichtung des Codes des Prototypen empfanden wir es letztendlich auch als sinnvoller, unseren Code komplett abzukapseln. Dieser kann im Anschluss an das Projekt erneut in den existierenden (und sich ebenfalls verändernden) Prototyp eingebracht werden. Die Softwareentwicklung dieses Projekts verfolgte somit zwei primäre Ziele:

1. Das Etablieren von grundlegender (objektorientierter) Design-Patterns als Gerüst für weitere Features
2. Grundlagen für geometrische Eingaben

Die etablierten Design-Patterns und Funktionalitäten werden im folgenden Abschnitt beschrieben; die entwickelten Features werden im Bereich 4 vorgestellt. Um die oben genannten Vorgaben zu erzielen, wurden einige Schwerpunkt im Entwicklungsprozess gesetzt. Erstens wurde darauf geachtet, möglichst wenige Bibliotheken und Pakete zu benutzen, um eine reibungsfreie Einbindung in das existierende Projekt zu ermöglichen. Aus der gleichen Überlegung wurde auch kein Code des existierenden Prototyps verwendet oder referenziert. Zudem wurde ein sehr großer Fokus auf sauberen Code und gute Programmierpraktiken gelegt, um eine möglichst gute Wartbarkeit und Erweiterbarkeit zu erzielen. Dementsprechend wurde die Balance zwischen Featureanzahl und -stabilität auch deutlich in letztere Richtung verlegt.

In dem Projekt wurde für Unity 2019.2 und anschließend **2019.3** programmiert. Der Code ist dementsprechend in C# 6 geschrieben und benutzt auch einige der neuen Sprachfeatures (wie *expression-bodied members, string interpolation* und *null propagation*). Es ist somit erforderlich, dass die Projekteinstellungen in Unity (unter den *Player*-Einstellungen) ein *Api Compatibility Level* von *.NET 4.X* eingestellt haben. Der Quellcode folgt den Standard-.NET-Konventionen[1] und benutzt eine Tab-Tiefe von zwei Leerzeichen.

Für die Beschreibung des Codes wird ein grundlegendes Verständnis der Unity Engine

[1] `https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/`, aufgerufen am 14.03.2020

sowie von C# vorausgesetzt.

Zusätzlich zu einem komprimierten Archiv im Anhang ist der Code auch auf dem Git-Repository verfügbar, welches als Versionskontrollsystem während der Entwicklung verwendet wurde[2].

5.1 Softwarearchitektur

Damit Softwareprojekte auch mit zunehmender Komplexität und mehreren Programmierern wartbar und erweiterbar bleiben, ist ein strukturiertes Vorgehen und Entwerfen des Codes erforderlich. Hierbei haben sich neben den groben Ansätzen (wie z. B. objektorientiertes und funktionelles Programmieren) auch sogenannte *Design-Patterns* herauskristallisiert. Das wegweisende Buch *Design Patterns* der sogenannten *Gang of Four*Gamma et al., 1995 war hierbei ein großer Faktor zur Etablierung und Standardisierung. Während des Software-Entwurfs ist es möglich, die Programmstruktur zu gestalten, in dem Design-Patterns als Gerüst und Bausteine verwendet werden. Insbesondere in Unity ist die disziplinierte Erstellung und Einhaltung von Design-Patterns wichtig. Aufgrund der Eigenheiten der Engine keinen zentralen Startpunkt bzw. Wurzel des Codes, wie eine *main()*-Methode. Stattdessen wird eine Aufteilung in dezentralisierte Bausteine (*MonoBehaviour*s) als primäre Entwicklungsstrategie nahegelegt. Dies ermöglicht zwar eine relativ flexible Komposition von Funktionen, wenn die *MonoBehaviour*s klein und modular gehalten werden und ist eine hervorragende Strategie für Prototypen. Für längerfristige Projekte werden jedoch folgende Aspekte schnell problematisch:

- Werden die *MonoBehaviour*s tatsächlich klein und modular gehalten, steigt deren Anzahl sehr schnell an. Zudem müssen diese *MonoBehaviour*s auch für viele Anwendungsfälle kommunizieren, sie sind also oft voneinander abhängig. Der hierfür gängige Ausdruck ist, dass sie *Dependencies* besitzen, d.h. dass sie von anderen Typen abhängig sind. Zwar ermöglicht Unity mittels *GetComponent()* einen bequemen Zugriff innerhalb des gleichen *GameObject*s, und mittels *FindObjectOfType* ein Auffinden innerhalb einer Szene, diese Methoden sind jedoch sehr fehleranfällig bei fehlenden Komponenten. Noch fehleranfälliger, aber leider ein bequemer und somit oft benutzter Weg, ist das Einsetzen von Dependency-Referenzen mittels der Benutzeroberfläche. Dies ist innerhalb des gleichen *GameObject*s (und als Verweis auf *Prefab*s) akzeptabel, ist jedoch hochproblematisch für Referenzen innerhalb der Szene und sollte minimiert werden.
- Erfahrungsgemäß häufiger werden *MonoBehaviour*s jedoch *nicht* klein und modular geschrieben. Sie werden oft nach und nach erweitert (und nicht refactored). Hierbei werden oft mehrere Funktionen übernommen, es werden z.B. Logik und Darstellung (*logic* und *view*) vermischt und verwoben, was Änderungen generell sehr fehleranfällig macht.
- Eine Aufteilung in *GameObject*s und *MonoBehaviours* ist zwar ein nützlicher Anfang,

[2] `https://gitlab.tubit.tu-berlin.de/rueger/AUT-UnityInterface.git`

aber stellt dennoch keine größere, semantische Struktur für Softwareprojekte da, die häufig in Aspekte wie Berechnungen *business logic* und Benutzeroberfläche *view* unterteilt werden, die idealerweise unabhängig voneinander änderbar sind (*loose coupling*).

Diese Probleme waren auch im existierenden Prototyp ersichtlich und haben nach unserer Einschätzung eine Erweiterung des bestehenden Codes verhindert. Stattdessen haben wir folgende Ansätze zum Umgehen der oben beschriebenen Problematik gewählt:

1. Anwenden des Model-View-ViewModel-Musters (MVVM).
2. Service-Locator-Muster als Alternative zu Inspektor-Referenzen und *FindObjectOfType*
3. Eine zentralisierte, State-Machine-artige Zustandsverwaltung.
4. Modularisierung und Verfestigung von Programm-Bestandteilen mittels *Prefabs*

5.1.1 Model-View-ViewModel

Bei MVVM handelt es sich um ein von Microsoft entwickeltes Konzept zur Aufteilung von Softwareprogrammen[3]. Das Kernprinzip ist das strikte Trennen des Codes in drei Kategorien:

- Model - beinhaltet Daten und sog. *business logic*. In unserem Beispiel sind dies Geometrie-Daten (Punkte, Strecken...) und geometrische Funktionalitäten (wie das Finden der nächsten Strecke zu einem gegebenen Punkt.) Bei dem Prototypen würde hierbei auch noch z.B. der Netzwerk-Code mit dem Roboter fallen.
- View - die Schnittstelle zum Benutzer, also sowohl die grafische (und akkustische) Ausgabe, als auch die Eingabemöglichkeiten.
- ViewModel - als "Vermittler" zwischen View und Model. Oft liegen Daten und Funktionen im Model nicht anzeigegerecht vor; z.B. wenn Einheiten umgerechnet werden müssen. Zudem werden auf dieser Ebene Informationen verwaltet und verarbeitet, die nicht zur *business logic* zählen - ein Beispiel hierfür ist das Aufbewahren von ausgewählten Objekten, das Filtern von Listen (die dem *Model* entspringen) etc.

Zusätzlich zu dieser Auftrennung ist noch ein weiterer Aspekt entscheidend: Die Kategorien dürfen nur begrenzt voneinander wissen. Dies kann man sich als "Kette" vorstellen, in dem das ViewModel nur das Model kennt (und von diesem abhängt) - und ebenso *View*-Objekte nur vom *ViewModel*. Hieraus ergibt sich auch, dass *View*-Objekte ihren Zustand aus dem *ViewModel* ablesen und diesen über dessen Schnittstelle modifizieren können. Das gleiche Verhältnis gilt für das *ViewModel* zum *Model*.

Nach Microsofts Beschreibung soll dieses Muster event-gesteuert sein, d. h. Änderungen werden mittels C#-*delegates* weitergeleitet. Zusätzlich wird kein Hardcoding des Views

[3] `https://docs.microsoft.com/de-de/xamarin/xamarin-forms/enterprise-application-patterns/mvvm` aufgerufen am 14.03.2020

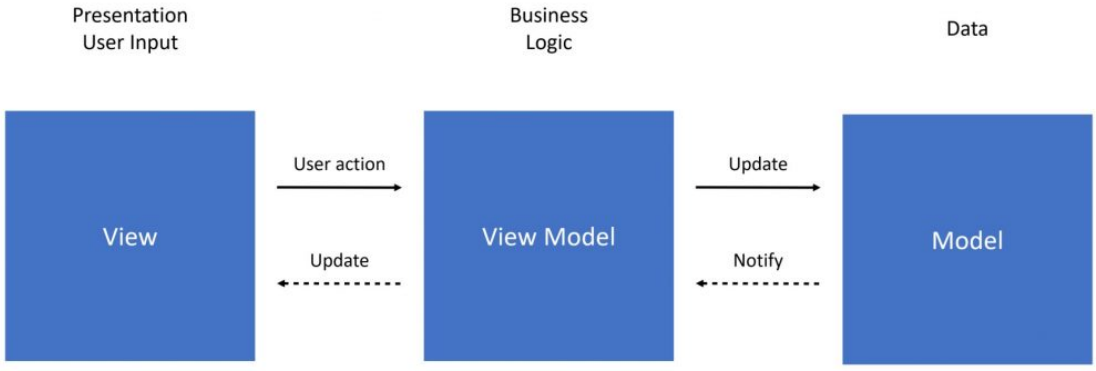

Abbildung 5.1: Visualisierung des MVVM-Musters, Quelle: `https://quickbirdstudios.com/blog/app-architecture-our-functional-mvvm-approach-with-rx/`

zum ViewModel, sondern dynamisches *data binding* vorgesehen. Da beides den Rahmen dieses Projekts sprengen würde (und zudem keine angemessenen Open-Source Data-Binding-Frameworks für Unity's UI-Bilbiothek vorhanden sind) wurde hiervon abgesehen.

5.1.2 Service Locator

Um die oben beschriebene Problematik der Dependency-Zuweisung zu lösen, wird das Konzept von Services und des Service-Locators in das Projekt eingeführt. Services sind hierbei Klassen, die generell nur eine Klasseninstanz erfordern und deren Funktionen von vielen verschiedenen Klassen genutzt werden. Das Service-Locator-Pattern wurde ursprünglich von Software-Architektur-Pionier Martin Fowler beschrieben[4]. Das grundlegende Konzept hierbei ist, dass der Service-Locator über eine Liste von vorhanden Services (also Objekte, die von anderen Objekten benötigt werden) verfügt. Diese Services werden vom Service-Locator aufgespürt oder ggf. auch erzeugt. Der Service-Locator ermöglicht es anderen Objekten, ihre Dependencies "abzuholen", anstatt sie selbst zu finden oder erstellen zu müssen - und somit das *Single-Responsibility*-Prinzip des objektorientieren Programmierens verletzen würden. Mittlerweile wird der *Service-Locator* als sogenanntes Anti-Pattern angesehen und sollte nach Meinung einiger Software-Architekten[5] durch *Dependency Injection* ersetzt werden. Wir halten jedoch Service-Locators für ein Projekt dieser Größe für eine angemessenere Option; zudem ist Dependency Injection ein nicht unbedingt intuitiv zu verstehendes Konzept und dementsprechend für Universitätsprojekte kaum ratsam.

[4] `https://www.martinfowler.com/articles/injection.html`, aufgerufen am 14.03.2020
[5] z.B. `https://freecontent.manning.com/the-service-locator-anti-pattern/`, aufgerufen am 14.03.2020

5.1.3 Zentralisierte Zustandsverwaltung

Ein gängiges Problem in Unity, insbesondere bei Benutzeroberflächen, ist das koordinierte (De-)Aktivieren von *GameObject*s. Hierüber können, je nach Anwendungszustand, z.B. einige UI-Elemente ein- oder ausgeblendet werden. Dies sollte jedoch nicht programmatisch *hardgecoded* werden, sondern in einem ähnlichen Rahmen wie die UI-Elemente selbst anpassbar sein. Aus dieser Überlegung heraus haben wir ein endlicher-Automat-artiges System entwickelt, über das Zustände definiert und zentral geändert werden können - und deren Änderungen automatisch an "Schalter" übertragen werden. Wichtig hierbei ist die Funktionalität, dass sowohl durch Code, als auch durch Konfiguration im Unity Editor Zustandsänderungen definiert werden können.

5.1.4 Prefabs

Da Unity's derzeitige UI-Bibliothek auf *GameObject*s basiert, ist eine Konfiguration von Objekten im Editor leider die primäre Methode, um Benutzeroberflächen zu entwickeln. Glücklicherweise existieren seit Version 2018.3 sogenannte *nested prefabs*: Es können mehrere Prefabs ineinander eingebettet werden und dennoch unabhängig voneinander geändert werden. Dies erlaubt ein modulares Erstellen von UI-Elementen, wie z.B. Knopf-Prefabs, die in verschiedenen Menü-Prefabs enthalten sind. Dies ermöglicht ein flexibles und vergleichsweise robustes System zum Erstellen verschiedener UI-Elemente.

5.2 Dokumentation

Dokumentation ist ein essentieller Aspekt von Softwareprojekten und trägt oft maßgeblich zu deren Erfolg oder Scheitern bei (Forward und Lethbridge, 2002). Dies ist insbesondere der Fall, wenn keine langfristige Betreuung durch die ursprünglichen Entwickler möglich ist, wie z. B. in Universitäten oder Forschungsorganisationen mit häufigen und projektbezogenen Personalwechseln. Unter diesem Blickwinkel wurde in diesem Projekt besonderer Wert auf eine angemessene Dokumentation gelegt. Hierfür benutzten wir einen dualen Ansatz, welcher die gängige Praxis im Feld und auch den Stand der Forschung widerspiegelt (Meng, Steinhardt und Schubert, 2018): Zusätzlich zu der konzeptbezogenen Fließtext-Dokumentation in diesem Dokument (und in ReadMe-Dateien im Projekt) wurde eine API-Dokumentation generiert, die einen Überblick und eine benutzergerechte Durchsuchung der Codestruktur erlaubt.

5.2.1 API-Dokumentation

Die API-Dokumentation wurde aus XML-Kommentaren innerhalb des Quellcodes mit Doxygen[6]erstellt. Die Darstellung auf HTML-Seiten ist etablierter Standard und wird in ähnlicher Form z.B. auch von Unity verwendet. Die generierte HTML-Dokumentation befindet sich als komprimiertes Archiv im Anhang. Um die Dokumentation (z. B. nach

Änderungen am Code) neu zu generieren, muss Doxygen heruntergeladen und installiert werden[6]. Anschließend kann Doxygen über die Kommandozeile ausgeführt werden. Hierbei muss als Argument der Pfad der Doxygen-Konfigurationsdatei des Projektes angegeben werden, welcher sich im eingereichten Repository unter *projectorPlacement/Docs/* befindet. Die generierte Dokumentation befindet sich dann im automatisch generierten Ordner *html*. Mit den eingereichten Einstellungen werden nur Quellcode-Dateien im Ordner *projectorPlacement/Assets/Scripts/* mit einbezogen und HTML-Output generiert. Die Doxygen-Konfigurationsdatei ermöglicht eine Anpassung vieler Parameter und kann mittels Texteditoren angepasst werden.

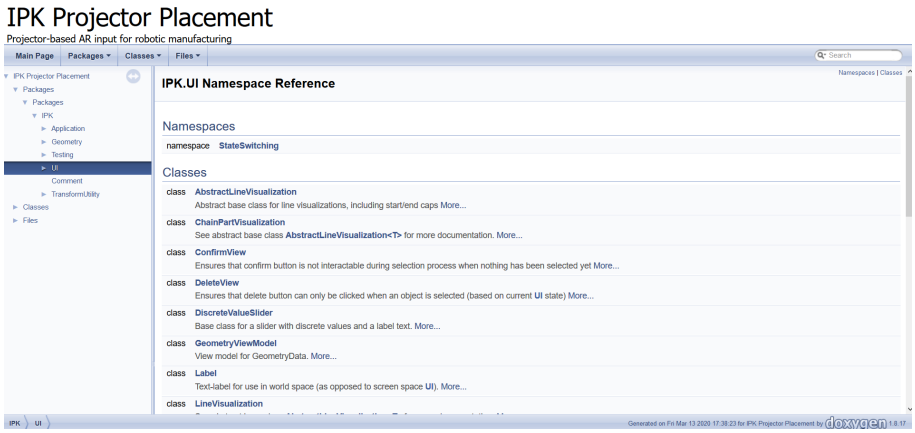

Abbildung 5.2: Beispiel-Screenshot der HTML-Dokumentation

5.2.2 Dokumentation im Code

Das Projekt folgt dem Ansatz des selbstdokumentierenden Codes: Durch aussagekräftige Variablen-, Klassen- und Funktionsnamen sollen der Großteil des Codes ohne zusätzliche Kommentare für andere Programmierende verständlich sein. Falls dies nicht möglich ist, sollen zusätzliche Kommentare nicht das *"was"*, sondern das *"warum"* beschreiben: Beispiele hierfür sind komplexe Funktionen, die auf mathematischen Sätzen beruhen oder auf den ersten Blick als unnötig komplex erscheinende, jedoch notwendige, *Workarounds*.

```
31  /// <summary>
32  /// Call this constructor if you do not expose the line visualization in the editor but directly supply the visualizaion prefabs.
33  /// Includes nitialization.
34  /// </summary>
35  /// <param name="segmentCapPrefab">May be null.</param>
36  /// <param name="parent">Generally the calling GameObject's transform. Line visualization is placed in scene root if null.</param>
37  public AbstractLineVisualization(GameObject chainVisualizationPrefab, GameObject segmentCapPrefab, Color defaultColor, Transform pa
44
45  public void SetActive(bool active)...
51
52  /// <summary> Creates visualization GameObjects in the scene.
56  public virtual void Initialize(Transform parent)...
67
68  /// <summary> Updates visualization - must be updated manually every frame.
71  public abstract void Update(T data, Color? colorOverride = default, float heightOffset = 0);
```

Abbildung 5.3: Beispielhafter Code-Abschnitt mit XML-Dokumentation

Zudem wurden öffentliche Schnittstellen und Klassen mit XML-Kommentaren versehen. Diese dienen einerseits der oben vorgestellten Dokumentationsgenerierung; andererseits werden sie auch durch Visual Studios *Intellisense*-Unterstützung eingeblendet. So wird

[6] siehe `http://www.doxygen.nl/`, aufgerufen am 14.03.2020

beispielsweise beim eintippen eines Funktionsaufrufs die kommentierte Beschreibung der Funktion angezeigt. Falls Funktionsparameter erforderlich (und dokumentiert) sind, wird auch für diese beim Eintippen eine Beschreibung angezeigt. Dies ermöglicht einen reibungsfreien Informationstransfer, ohne die Dokumentaiton in einer externen Anwendung aufrufen zu müssen.

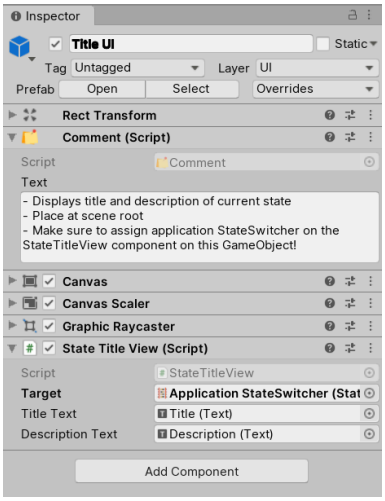

Abbildung 5.4: Dokumentation von GameObjects mit Kommentar-Component

Bei Unity-Projekten sind jedoch auch zusätzlich Szenen und Konfigurationen von *GameObject*s wichtig und müssen dementsprechend nachvollziehbar kommuniziert werden. Hierzu wurde ein einfacher Kommentar-Script erstellt, der an *GameObject*s angefügt werden kann und Hinweise zu deren Funktion und Abhängigkeiten liefert.

5.3 Liste der Erweiterungen und Anpassungen im Rahmen des Projektes

Neben der Auflistung des neu erstellten Codes in der API-Dokumentation soll diese Liste eine semantische Übersicht über die erarbeiteten Bestandteile geben. Folgende Features wurden erstellt:

- Input: Der Finger-Input des Prototypen war auf dem Physik-System basierend und nur umständlich an PCs zu bedienen. Um bessere Test- und Iterationsmöglichkeiten zu schaffen, wurde ein neues Input-System geschrieben, das auf Grundlage von Unitys Input-System funktioniert. Hierüber ist eine nahtlose Bedienung von Unitys UI-Bibliothek möglich; darüber hinaus kann somit auch an PCs mit ein regulären Maus getestet werden.
- State Switching: Hierbei handelt es sich um die Implementierung der oben beschriebenen zentralisierten Zustandsverwaltung. Neben der Grundfunktion des Zustandsumschaltens lassen sich zudem mehrere Zustände gleichzeitig verwalten, mittels sogenannter *Contexts*, die dem zentralen *StateSwitcher* und den dazugehörigen verteilten *StateSwitch*-Instanzen zugewiesen werden. Mittels des StateSwit-

cher kann der Zustand gewechselt werden, welches wiederum an die StateSwitch-Instanzen mit dem gleichen Context weitergeleitet wird. Diese schalten dann, je nach Zustand und Konfiguration, ihr GameObject an oder aus. Zudem lassen sich auch Abfolgen von Zuständen mittels einer *StateSequence* definieren und abspielen.

- Service-Locator: Implementierung des oben beschriebenen Konzepts, inklusive *ILocatableService*-Interface, welches von Service-Typen zu implementieren ist. Funktioniert nur für *MonoBehaviour*-Typen.
- Ein zentralisiertes Audio-System, damit nur AudioClips zum abspielen von Tönen erforderlich ist, bzw. im Fall von UI-Tönen lediglich ein enum-Wert. Dies reduziert den Erstellungsaufwand zum abspielen von Tönen enorm und eliminiert *Boilerplate*-Code.
- Erstellen verschiedener modularer UI-Prefabs
- Kettenzüge: Verschiedene Klassen und Interfaces für Kettenzüge, jedoch bisher nur für Liniensegmente.
- Punkte: Definition von Punkten, sowie des *IReferenceSystem*-Interface, über das Objekte als Koordinatensystem fungieren können (z. B. Liniensegmente.) Punkte müssen relativ zu einem IReferenceSystem definiert werden.
- Model und ViewModel für die Geometriebearbeitung
- Eine Vielzahl an Views für die Darstellung von den Werkstück-Kanten, sowie dem Modifizieren und Erstellen von Punkten und Kettensegmenten. Dies ist der Hauptanteil des Codes und wird detaillierter in 4 sowie der API-Dokumentation beschrieben.

5.4 Erweitern und ändern des Projektes

Die einfachste Methode um das Projekt zu erweitern, ist wohl die Szene *PolyChainTest* als Kopie abzuspeichern und zu modifizieren. Alternativ können auch einzelne Prefabs benutzt werden, wobei jedoch darauf geachtet werden muss, dass die erforderlichen Dependencies (welche sich in der Regel im Application-Prefab) existieren. Zudem muss ggf. der Anwendungs-StateSwitcher bei manchen MonoBehaviours zugewiesen werden.

5.4.1 Neue Anwendungsfunktionen

Sollten neue Anwendungsbestandteile hinzugefügt werden, sollte (neben der Modularisierung im Code) auch darauf geachtet werden, dass die dazugehörigen GameObject-Konfigurationen als Prefabs abgespeichert werden. Diese sollten möglichst alleinstehend funktionieren, also Plug-&Play-mäßig in Szenen einfügbar sein, bzw. nur das Application-Prefab erfordern[7].

[7] Alle Prefabs des Projektes befinden sich im Resources-Ordner.

5.4.2 Benutzen des Service Locators

Hilfreich bei der Modularisierung (und zur Minimierung von Setup-Arbeit im Inspector) ist der Einsatz des Service Locators. Sofern Services nur benötigt werden, kann eine Service-Instanz einfach mittels *ServiceLocator.Instance.GetService<T>();* angefordert werden, wobei *T* der erforderliche Service-Typ ist. Wichtig hierbei ist, dass dieser Aufruf frühestens in Start() erfolgt, da erst dann sichergestellt ist, dass alle Services gefunden wurden. Ein Anwendungsbeispiel hierfür findet sich in *PlayUIAudio.cs*. Natürlich können auch neue Services erstellt werden - hierbei bieten sich Klassen an, die in vielen verschiedenen MonoBehaviour-Instanzen benötigt bzw. referenziert werden, von denen aber gleichzeitig nur eine Instanz existiert (oft auch Singletons genannt.) Diese Services sollten von MonoBehaviour erben und zudem die Schnittstelle *ISingletonService* aus dem Namespace *IPK.Application* implementieren. Hierzu reicht es, den implementierenden Typ anzugeben, unter dem der Service-Typ gefunden werden kann - in der Regel ist das einfach der Typ/die Klasse des Services. Damit der Service lokalisiert werden kann, muss eine Instanz des neu erstellten Typs in der Szene vorhanden sein - er wird dann automatisch vom ServiceLocator gefunden, natürlich nur falls eine ServiceLocator-Instanz in der Szene vorhanden ist.

5.4.3 State Machine

Vor allem für das Erstellen von User-Interfaces kann der StateSwitch eingesetzt werden. Prinzipiell kann aber hiermit jegliche MonoBehaviour-basierte Funktionalität zustandsbasiert an- oder ausgeschaltet werden. Wie oben erwähnt steuert der Context, welche StateSwitcher und StateSwitche miteinander kommunizieren. Im jetzigen Anwendungszustand existiert nur ein *Context*, der Application-Context. Sollen also andere Zustände als die der Benutzeroberfläche verwaltet werden, sollte ein zusätzlicher Context erstellt werden (Rechtsklick im Project View und dann Create->IPK->UI->Context.) Dieser ist dann den entsprechenden StateSwitcher und StateSwitch-Instanzen zuzuweisen. Natürlich lassen sich auch neue States und StateSequences nach Belieben definieren.

5.4.4 Interaktive UI

Hierfür ist primär an Unitys offizielle UI-Dokumentation zu verweisen, insbesondere die Abschnitte zu Unity-Events. Mit diesen kann dynamisch im Inspektor Funktionalität zugewiesen werden, ohne hierfür Code schreiben zu müssen. Vor allem sind hierbei *Wrapper-MonoBehaviour* nützlich, welche Parameterkonfigurationen erlauben, den Methodenaufruf dann aber mittels einer parameterlosen Methode für Unity-Events bereit stellen. Ein Beispiel hierfür ist *PlayUIAudio.cs*. Hierüber lassen sich auch alle StateSwitch-Funktionalitäten bedienen, wovon in vielen UI-Elementen gebrauch gemacht wird.

5.4.5 Audio

Mittels des AudioPlayer-Services ist es möglich mit nur einer Zeile Code Sounds auszugeben. Ein Sonderfall hierbei ist der Service *UIAudio* - in diesem werden UI-Töne als enum definiert. In der UIAudio-Instanz (am Application prefab) werden den enums dann AudioClips zugewiesen werden. Somit können UI-Töne mit Hilfe der UIAudio Service-Instanz in Code ausgelöst werden, oder aber mittels einem PlayAudioComponent über Unity-Events gestartet werden, z.B. bei Button-Clicks. Um neue Audio-Töne hinzuzufügen, kann einfach der enum (am unteren Ende!) erweitert werden. Anschließend muss ein entsprechender AudioClip in der UIAudio-Instanz zugewiesen werden.

5.4.6 Erstellen eines neuen UI-Bestandteils

Um ein neues UI-Objekt zu erstellen kommen alle oben genannten Aspekte zusammen - zudem sollte auch das MVVM-Muster angewendet werden. Für ein grundlegendes Verständnis ist es empfehlenswert, die existierenden UI-Bestandteile in der PolyChainTest-Szene zu analysieren; hierüber sollten Grundlagen und Muster erkennbar werden, wie mit den o.g. Funktionen effektiv UI-Objekte erstellt werden können. Grundlage der meisten UI-Objekte ist ein View-MonoBehaviour: Dieses stellt Funktionalität und Daten vom ViewModel bereit und steuert ggf. Eingabe- und Darstellungslogik. Auch hierfür ist es empfehlenswert, den Code einiger existierende View-Klassen zu analysieren. Für die tatsächliche Darstellung können entweder Szenenobjekte oder Unity UI-Objekte verwendet werden. Ersteres wird in der bestehenden Anwendung z.B. für Segmentdarstellungen verwendet und beruht meist auf Prefabs, die in der Laufzeit instanziiert werden. UI-Objekte sollten hingegen bereits vollständig zusammengefügt werden und in Prefabs im Projekt abgespeichert werden. Hierbei sollten grundlegende Elemente wie Buttons auch in Form von Prefabs verwendet werden, um über eine einfache Möglichkeit zur einheitlichen Gestaltung zu verfügen. Meist ist es auch sinnvoll, die Funktionalität in Form eines sog. *Widgets* zusammenzufassen, die in verschiedenen Menüdarstellungen eingefügt werden kann. Sowohl für einzelne UI-Objekte wie *Buttons*, als auch für komplexere *Widgets*, ist ein effektiver Einsatz von Unitys *Nested Prefabs* und *Prefab Variant*-Funktionalitäten wichtig. Hierfür ist auf deren offizielle Dokumentation zu verweisen.

5.5 Limitationen & Erweiterungsvorschläge

Da dieses Projekt in einem zeitlich beschränkten Rahmen stattfindet, wurden einige Features nicht implementiert, die wünschenswert und/oder naheliegend gewesen wären. Diese sollen hier als Ausblick für die Zukunft aufgelistet werden, inklusive einer kurzen Beschreibung, wie diese Features in den existierenden Code eingebettet werden können:

- **Integration mit anderen Teilen des Projektes:** Aufgrund oben beschriebener Gründe wurde keine Integration mit z.B. der Kommunikation mit dem Roboter vorge-

nommen.

- **Visualisierung des (Finger-)Inputs:** Für das Hovern kann hierfür der Effekt des Prototypen mit der Funktion *InputUtility.GetFingerPosition()* übernommen werden. Zusätzlich sollte jedoch eine Visualisierung erstellen werden, die anzeigt ob der Timer für die Tap-Detektion läuft. Dies kann z. B. mit einem sich fülllenden Kreis realisiert werden. Hierzu eignet sich ein World-Space UI-Canvas (aus Unity UI-Bibliothek). Dieser hat ein Image-Objekt mit einer Kreistextur, dessen Füllungsmodus auf *radial* gesetzt wird, und dessen Füllungswert an den Timer der Tap-Erkennung gekoppelt wird. Die Position des World-Space-Canvas ist ebenfalls über *InputUtility.GetFingerPosition()* abrufbar.
- **Rückgängig-Funktion:** Als Grundlage einer guten User-Experience sollte die Möglichkeit gegeben werden, Aktionen und Fehler rückgängig zu machen. Dies ist umsetzbar, indem das *Model* (also *GeometryData*) serialisiert wird - dies ist mit Unitys eingebautem JSON-Serialisierer möglich. Nach jeder Zustandsänderung wird das Model serialisiert. Eine Rückgängig/Wiederholen-Funktion ist mittels zweier *Stack<byte[]>* möglich. Ein Problem hierbei ist jedoch die Auswahl der Zeitpunkte, zu denen der Zustand des Models serialisiert werden soll - so sollte z.B. beim Verschieben eines Punktes nicht zu jedem Frame serialisiert werden, sondern nur der Zustand *nach* dem Verschieben.
- **Zentralisierter/verbesserter Zugriff auf StateSwitcher:** Referenzen zum *StateSwitcher* sind die einzigen Referenzen innerhalb der Szene, die jedoch (wie oben dargestellt) sehr fehleranfällig sind. Es wäre somit wünschenswert, den StateSwitcher ebenfalls über den ServiceLocator zugänglich zu machen. Jedoch ist hierbei noch ein weiterer Parameter entscheidend: Obwohl nur ein *StateSwitcher* pro *Context* existieren sollte, können mehrere Contexts und somit auch mehrere StateSwitcher existieren. Somit müsste wahrscheinlich der ServiceLocator angepasst werden, damit dieser einen bequemen aber dennoch korrekten/robusten Zugriff auf die StateSwitcher ermöglicht.
- **Reihenfolge von Punkten/Abschnitten:** Da letztendlich das Ziel ein Pfad für den Endeffektor eines Roboters ist, ist es zudem wichtig, die Abfahr-Reihenfolge der Punkte und Abschnitte zu ermöglichen. Punkte und Abschnitte werden bereits in (Reihenfolge-enthaltenden Listen gespeichert und das Austauschen an zwei Indizes ist trivial. Dementsprechend ist hierbei die größte Herausforderung der Entwurf der Benutzeroberfläche. Ein möglicher Ansatz wäre hierbei die Auswahl eines einzelnen Elements und eine Sortierung mittels eines Auf-/Ab-Pfeils. Eine Alternative hierzu wären z.B. sortierbare Listen[8].
- **Visualisierung des Roboterpfades:** Als Erweiterung des vorherigen Punktes könnte auch der Roboterpfad visualisiert werden, z. B. durch eine Linie, oder durch ein virtuelles Abfahren mittels eines Punktes oder Kreuzes.
- **Einschränken des Segmenterstellung:** Für die Erstellung von Schweißnähten soll-

[8] `https://forum.unity.com/threads/free-reorderable-list.364600/`

te die Segmenterstellung auf nur einen gewissen Teil einer Werkstückkante begrenzt werden. Hierbei muss vor allem beachtet werden, dass diese Teil-Abschnitte wahrscheinlich nicht geschlossen sind, und somit eine Invertierung des Segments nicht möglich ist.

- **Kreissegmente:** Das Interface IPolyChain kann prinzipiell beliebige ISegment-Typen enthalten. Jedoch ist bisher LinearSegment der einzige implementierte ISegment-Typ. Die Implementierung von Kreissegmenten müsste dementsprechend auch ISegment implementieren, ggf. muss auch LineSegmentChain minimal angepasst werden, um auch Kreissegmente zu unterstützen.
- **Mehrere Werkstücke:** Hierzu müssen nicht nur die Werkstücke im Model und ViewModel verwaltet werden, sondern auch durch den Benutzer auswählbar sein. Zudem muss sichergestellt werden, dass alle Benutzereingaben auf das entsprechend ausgewählte Werkstück bezogen werden. Wahrscheinlich bietet es sich an, eine *GeometryData*-Instanz pro Wekstück zu verwalten.
- **Abspeichern von Plänen:** Grundlegend ist dies nur möglich, wenn a) entweder nur ein Werkstück oder b) die gleiche relative Anordnung mehrerer Werkstück sicher gestellt werden kann. Darüber hinaus müssten Identifikatoren für ein Werkstücke generiert werden, die zwischen der Werkstückerkennungs-Anwendung und der Unity-Anwendung konsistent sind. Die Identifikatoren sind anschließend im *GeometryData*-Model zu speichern. Das Model selbst ist relativ einfach speicherbar, mittels Unitys JSON-Serialisierer (siehe Punkt “Rückgängig-Funktion”.)

Als weitreichender Ausblick ist auch das Hinzufügen und Bearbeiten von Prozess-Parametern notwendig. Hierfür müsste sich zuerst überlegt werden, wie Prozess-Parameter (wie z.B. Bohrtiefe oder Schweißnahtdicke) an die Geometriedaten im Model gebunden werden können. Darüber hinaus sind jedoch grundlegende Usability-Überlegungen anzustellen, wie Prozessparameter sinnvoll Geometrieelementen (in der Benutzeroberfläche) zuzuordnen, zu erstellen und zu bearbeiten sind.

6 Usabilitytestung

An dieser Stelle gilt es die erstellte Konstruktion bezüglich ihrer Usability zu untersuchen. Dafür werden vorerst der Fokus, die Art und der Aufbau der Testung erläutert und im Anschluss die Testung inklusive Beschreibung und Ergebnissen betrachtet.

6.1 Einschränkung der Testung

Eine Limitation der Usabilitytestung ist, dass die Anwendung mittels Tablet und nicht direkt mittels AR Anwendung getestet werden konnte. Dies liegt an verschiedenen Faktoren. Ein Aspekt ist die Rekrutierung der zu testenden, welche deutlich erschwert werden würde, da die AR-Umgebung recht weit entfernt vom Hauptcampus liegt. Zudem kann angenommen werden, dass die anderen Maschinen in diesem Raum die Probanden potentiell von der Hauptaufgabe ablenken könnte. Des Weiteren kam die Sorge auf, dass die Usability der neu erstellten Anwendung schwer zu differenzieren sein könnte von der vorherig programmierten Umgebung, die es nicht zu testen gilt. Anhand des Vorliegens eben genannter Störfaktoren, wurde sich entschieden lediglich die erstellte Teilanwendung auf einem Tablett zu untersuchen. Zu beachten ist dabei, dass es sich sowohl bei AR-Umgebungen als auch bei Tablettnutzungen um gestenbasierte Eingaben via Finger handelt. Es wird sich von der Forschungsgruppe erhofft, dass diese Gemeinsamkeit eine Verallgemeinerung von Toucheingaben im Tablett zu Zeigeeingaben in der AR-Umgebung i.w.S. zulässt.

6.2 Fokus der Testung

Die folgende Usabilitytestung bezieht sich auf mehrere Usabilityaspekte und die diesbezügliche User Experience. Wichtig ist es dabei, Problemstellen herauszufiltern. Dabei gilt es Aspekte wie die Menüführung, Menüaufbau, Anleitungsdarstellungen und Funktionalitäten zu analysieren. Explizit gilt es zu ermitteln, als wie konsistent bestimmte Anwendungsprozesse empfunden werden. Des Weiteren wird bewertet, ob der Nutzer genügend informatives Feedback erhielt und der aktuelle Status des Programmes an diesen gut weitergeleitet wird. Ebenso erhoben werden das Kontrollgefühl und die Erwartungen des Anwenders. All diese Aspekte sind essentiell für eine optimale Nutzung und werden deshalb je Aufgabe abgefragt. Selbstverständlich wird zusätzlich erhoben, wie gut gewisse Aspekte der Anwendung ankamen.

6.3 Stichprobe

Aquiriert und befragt wurden insgesamt 5 Personen. Die Grunddaten der Stichprobe befinden sich der Übersichtlichkeit halber in Abbildung 6.1.

Alter	Geschlecht	Studium	Fachgebiet	Usabilityerfahrung	Schweißerfahrung
MW	Männlich	Student	Techn./Inf.	Ja	Ja
25.6	3	5	2x BA Informatik 1x BA Physik	2	0
SD	Weiblich	Kein Student	Nicht s.o.	Nein	Nein
2.07	2	0	2x MA Human Factors	3	5

Abbildung 6.1: Stichprobenkennwerte - Tabellarisch

Dabei wurde explizit darauf geachtet, dass die zwei Grundgruppen - angehende Experten aus dem technischen und Human-Factors-Bereich - befragt wurden.

6.4 Beschreibung der genutzten und erstellten Erhebungsgrundlage

Es wurde sich dafür entschieden, keine standardisierten Fragebögen oder ähnliches zu verwenden. Somit wird sich erhofft, spezifische Problematiken gezielter aufdecken zu können. Dennoch wurde sich auf die erwähnten Konzepte in Abschnitt 2.4 bezogen. Dabei wurden den Teilnehmern in der Testung verschiedene Aufgabenstellungen aufgegeben, welche umzusetzen sind. Diese dienen dazu, alle Funktionen von Interesse beobachten zu können und im Anschluss Fragen zu genau dem jeweiligen Aspekt der Anwendung stellen zu können. Insgesamt gab es zehn Aufgaben, welche teils aufeinander aufbauend gestaltet wurden. Daraufhin erfolgte eine Endbefragung. Zusätzlich wurden gewichtige Anmerkungen mit beachtet, welche im folgenden als Zitate erwähnt werden.

6.4.1 Beschreibung der Fragen der Aufgaben

Nach jeder Aufgabe wurden fünf Fragen gestellt, welche auf einer Skala von 1-5 beantwortet werden konnten. 1 steht dabei für “Nein, stimme gar nicht zu” , 2 für “nein, stimme nicht zu”, 3 für “ich weiß es nicht/neutral”, 4 für “ja, stimme zu” und 5 für “ja, stimme sehr zu”. Alle Fragen sind dabei so formuliert, dass der Wert ist, der am positivsten zu bewerten ist.

Fragen	Usabilityüberlegungen
1. “Hast Du die Interaktion als konsistent empfunden?”	Konsistenz ist die 1. Shneidermansche Regel siehe Abschnitt 2.4, welche besagt dass die konsistente Gestaltung von Eingaben, Layout usw. die Usability erhöht. Demnach sind höhere Werte als besser anzusehen.

2. "Hat das System Dir genügend informatives Feedback über deine Eingaben geliefert?"	Informatives Feedback ist die 3. Shneidermansche Regel siehe Abschnitt 2.4. Dieses bezieht sich darauf, dass der Nutzer ausreichend hilfreiches Feedback bekommt wie beispielsweise eine Bestätigung, dass eine Eingabe angenommen wurde. Auch das 5. Dialogprinzip Steuerbarkeit beschriebt diesen Aspekt, da es dem Nutzer ermöglicht werden soll, mittels Dialog Aktionen einleiten, lenken und beeinflussen zu können.
3. "Hat das System Dir ausreichend Informationen über seinen Status geliefert?"	Diese Frage bezieht sich auf mehrere Shneidermansche Regeln: • 3. Regel - Informatives Feedback - Ein aktueller Status ekann auch z.B. Informationen über erfolgte Eingaben etc. oder auch Informationen über das aktuelle Menü/Modus liefern. • 7. Regel - Benutzerkontrolle gewährleisten - Ein aktiver Status ist notwendig, um Entscheidungen zu treffen und zu planen. Zudem braucht der Nutzer aktuelle Informationen um Kontrolloptionen wahrnehmen zu können. • 8. Regel - Kurzzeitgedächtnis entlasten - Dem Nutzer wird mit einer guten Statusanzeige geholfen, den Überblick zu behalten, wodurch das Kurzzeitgedächtnis weniger beansprucht wird. Zudem kann sich auch hier auf das 5. Dialogprinzip Steuerbarkeit bezogen werden.
4. "Hast du das Gefühl gehabt, die Kontrolle über das System gehabt zu haben?"	7. Regel - Benutzerkontrolle gewährleisten - Wie eben beschrieben, spielt das Kontrollgefühl eine wichtige Rolle für den Nutzer bezüglich der wahrgenommenen Usability. Wie Frage 3. kann sich hier wieder auf das. 5. Dialogprinzip Steuerbarkeit bezogen werden. Hierbei scheinen auch die Prinzipien der Informationsdesign zu greifen, da sowohl Lesbarkeit, Unterscheidbarkeit als auch Erkennbarkeit eine wichtige Rolle in der gefühlten Kontrolle spielen.

5. "Hat die genutzte Funktion so funktioniert wie Du es erwartet hast?"	Hierbei wird sich auf die Dialogprinzipien bzgl. 3. Erwartungskonformität bezogen, welche sich damit befasst, Nutzern die selbstausgelöste Aktion unmittelbar anzuzeigen im jeweiligen Kontext. Dies ermöglicht diesem, passend und kontrolliert zu handeln.

Tabelle 6.1: Fragen bezüglich der Testaufgaben

6.4.2 Beschreibung der Endbefragung

Neben den eben erläuterten Fragen zu den Teilaufgaben, erfolgte eine Endbefragung bezüglich weiterer Kriterien. Diese beinhaltete sechs Fragen:

Fragen der Endbefragung	Usabilityüberlegungen
1. Frage "Was hältst du von der Lage des Menüs?"	Siehe Abschnitt 2.4 bezieht sich diese Frage etwas auf die Prinzipien der Informationsdesigns, wobei es nötig wäre auf Erkennbarkeit (1.) und Lesbarkeit (3.) zu achten. Da sowohl die Nutzeraufmerksamkeit auf die gewünschte Information (Eingabeelement des Interesses) als auch auf eine lesbare und verständliche Darstellung zu achten ist. Dies ist eher im weiteren Sinne auf die Lage zu beziehen, wobei diese dennoch die Lesbarkeit und Erkennbarkeit beeinflussen kann, da z.B. damit festgestellt werden kann, ob die rechte Lage des Menüs mit den zu erfüllenden Aufgaben zusammenpasst oder nicht.
2. Frage "Wie verständlich waren für dich die Überschriften der Teilfunktionen? (1-5)"	Auch hier ist ein Bezug zu den Designprinzipien zu erkennen v.a. zu 3. Lesbarkeit bzw. eine leicht lesbare und verständliche Informationsdarstellung.
3. Frage "Wie verständlich waren für dich die Buttonbezeichnungen? (1-5)"	Ein Bezug zu den Designprinzipien ist zu erkennen v.a. zu 3. Lesbarkeit bzw. eine leicht lesbare und verständliche Informationsdarstellung.
4. Frage "Wie verständlich waren für dich die Anweisungen (oben mitte)? (1-5)"	Abermals betrifft dies die Designprinzipien v.a. zu 3. Lesbarkeit bzw. eine leicht lesbare und verständliche Informationsdarstellung.
5. Frage "Wie gut fandest du die Menüführung? (1-5)"	Dies betrifft eher allgemeine Funktionalitäten und demnach ist dies eher eine allgemeine Frage bezüglich der Anwendung und dem Gefühl des Nutzenden währenddessen.

6. Frage "Hast du die Implementierung der Funktionen zum Bearbeiten der Punkte und Abschnitte als konsistent empfunden?"	Hier wird sich wieder auf die 8 Shneidermanschen Regeln bezogen, wobei die 1. Regel Konsistenz betrachtet wird. Nach der Erledigung aller Fragen, ist es möglich für den Testenden eine allgemeine Einschätzung abzugeben bzgl. z.B. der verschiedenen Löschfunktionen und deren Konsistenz.

Tabelle 6.2: Endbefragung der Testung

6.4.3 Kommentare während der Testung

Während des Testung wurde keine Video- oder Tonaufnahme erstellt und demnach kein Transkript erstellt. Stattdessen wurden tatsächlich lediglich die Aufgaben erklärt und anschließend bewertet. Damit dennoch qualitative Daten mit beachtet werden konnten, wurden einschlägige Aussagen der Testenden festgehalten. Auf diese gilt es sich zudem in der Auswertung zu beziehen. Dabei wurden diese entweder den zugehörigen Aufgaben zugeordnet/währenddessen notiert, oder nach der Endbefragung abgegeben. Dadurch wird ermöglicht allgemeine Thematiken herauszustellen. Vermutet werden dabei, dass weitere Punkte der Usability wie nach den 8 Shneidermanschen Regeln auftauchen werden, da beispielsweise keine "Zurück"-Funktion erstellt wurde, welche nach 6. Umkehrbarkeit (gemachte Eingaben rückgängig machen) wichtig für viele Anwender zu sein scheint. Da derartige Funktionalitäten aufwendig sind und es möglich ist beispielsweise Strecken zu löschen und mit minimalem Aufwand neu zu erstellen, wurde sich im Rahmen dieses Projektes gegen solche Funktionen entschieden.

6.5 Aufbau der Testung

Der Aufbau der Testung bezieht sich auf das Setting in dem diese stattfindet. Herbei handelte es sich um eine Testung, welche in einer ruhigen, abgeschirmten Umgebung durchgeführt werden sollte. Hierfür wurde ein Raum nur für die Testung des Human Factors Bereiches genutzt. In welchem sich nur die Testperson und die beiden Versuchsleiter aufhielten. Die Versuchsperson wurde an einen leeren Schreibtisch gesetzt. Alle Befragungen wurden von zwei Versuchsleitern zusammen durchgeführt. Dabei hatte einer die Funktionalität, der Versuchsperson die Aufgaben zu erklären und eventuell einzugreifen. Diese war somit verantwortlich für die Hauptkommunikation. Der zweite Versuchsleiter hatte vor allem die Aufgabe zu protokollieren und gegebenfalls nachzuhaken, um Aussagen klarzustellen, um diese eindeutig verständlich festzuhalten. Der kommunizierende Versuchsleiter wurde während der Testung neben die Versuchsperson gesetzt. Der Protokollant befand sich hinter dieser. Zusätzlich wurde der Testperson ein Schreibblock mit einem Stift zur Verfügung gestellt, um etwaige Probleme visuell darstellen zu können. Eben beschriebenes ist zudem in Abbildung 6.2 dargestellt.

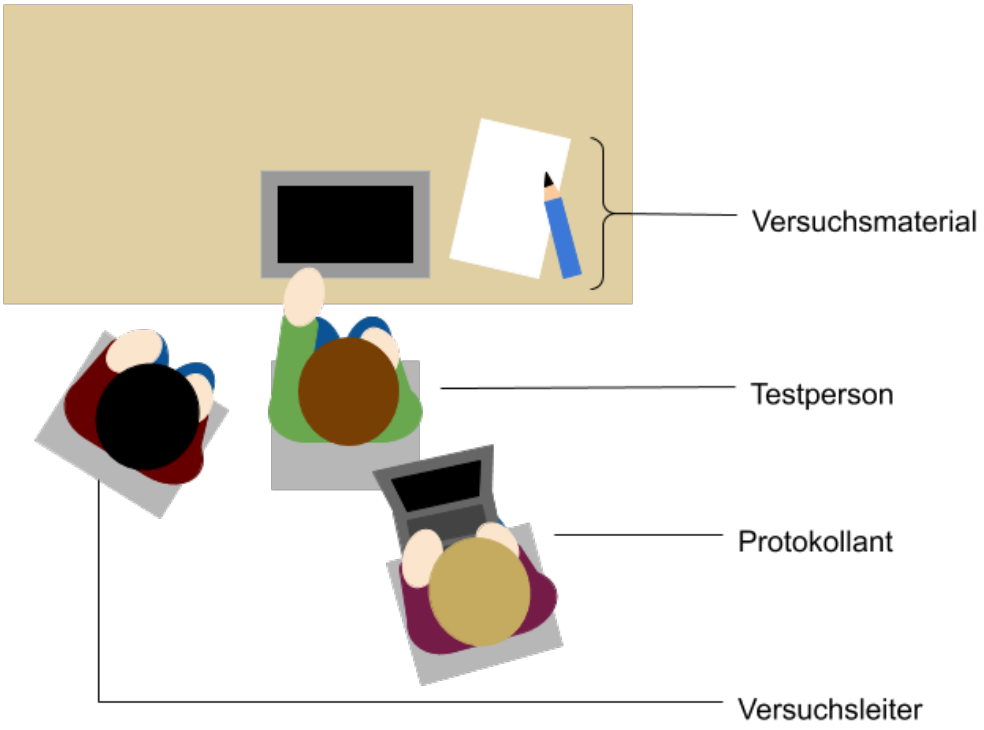

Abbildung 6.2: Aufbau / Setting der Testung

6.6 Ablauf der Testung

Die Testung wurde, wie in Abbildung 6.3 chronologisch dargestellt, durchgeführt. Dabei wurde mit einer Begrüßung begonnen, woraufhin demographische Fragen gestellt wurden, welche der Stichprobenbeschreibung dienen. Daraufhin wurden alle zehn Aufgaben abgearbeitet. Am Ende erfolgte die Endbefragung und die Verabschiedung.

Im Folgenden werden die Aufgaben und standardisierten Anweisungen der Versuchsleiter angegeben. Nicht angehängt werden die Belehrung bezüglich Anonymisierung u.ä., da es sich bei diesen um allgemein geläufige Vorgänge handelt.

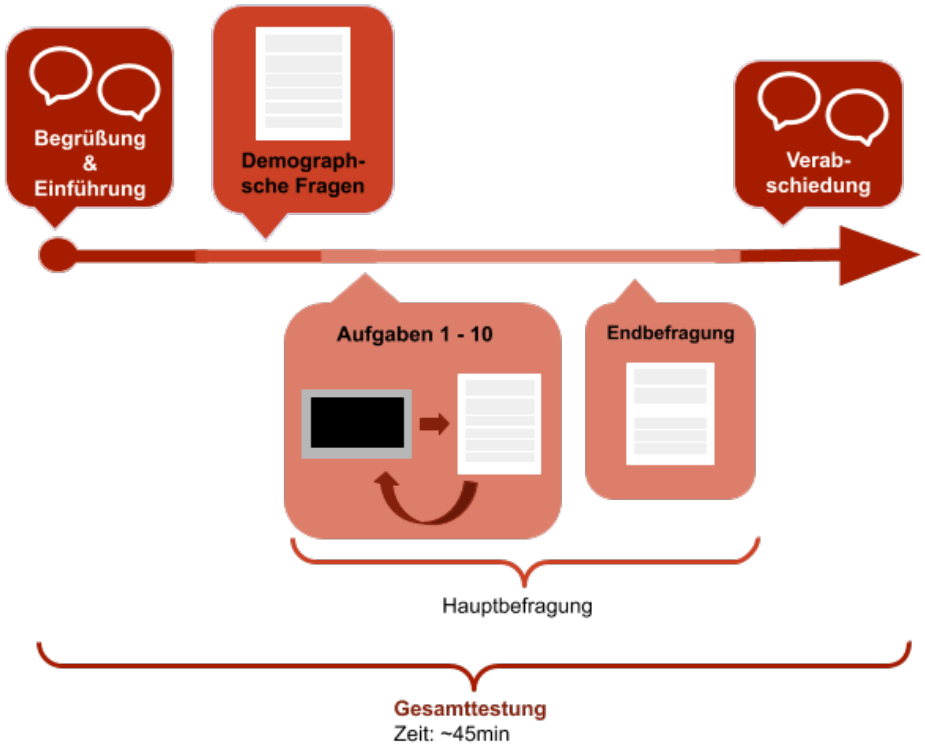

Abbildung 6.3: Zeitstrahl - Ablauf der Testung

Testungsablauf	Anweisungen
Begrüßung	Im Folgenden werde ich Dir eine Touchapplikation vorstellen zu welcher Dir Aufgaben gestellt werden. Vorerst möchte ich allerdings erklären, welchen Zweck diese zu erfüllen hat. Es handelt sich dabei um beispielsweise Schweißvorgänge. In der App, die du gleich testen wirst, werden diese simuliert. Du wirst eine Form sehen, welche ein Bauteil darstellt. Ziel ist es, an diesem ein Segment z.B. eine Schweißnaht festzulegen. Man könnte somit mittels Zeigen des Fingers bestimmen, wo das Bauteil beispielsweise an einem anderen Bauteil zu verbinden ist. Du wirst dafür gleich die Simulation testen. In der Realität würde man allerdings, ein richtiges Bauteil vor sich haben und die Handbewegungen würden mittels Sensoren/Kamera nachverfolgt werden. Dies dient lediglich als Information, da das Menü dementsprechend gestaltet ist, dass es gut in einer derartigen Umgebung zu nutzen ist.

Einführung	Auf diesem Tablett siehst du die erstellte Applikation. In der Mitte siehst du eine Form. Diese stellt das Bauteil dar. (Kanten mit Finger zeigen). Rechts befindet sich das Menü. Nutze dieses, um die folgenden Aufgaben zu lösen. Die folgenden Fragen werden auf einer Skala von 1-5 bewertet. 1 steht dabei für "Nein, stimme gar nicht zu" , 2 für "nein, stimme nicht zu", 3 für "ich weiß es nicht/neutral", 4 für "ja, stimme zu" und 5 für "ja, stimme sehr zu".
1. Aufgabe	Deine erste Aufgabe ist es, einen Punkt ca. in der Mitte des Objektes zu erstellen. Verwende dafür die verfügbaren Funktionen im Menü. Bei Fragen kannst du dich an deinen Versuchsleiter wenden.
2. Aufgabe	Lösche nun den eben erstellten Punkt.
3. Aufgabe	Erstelle nun erneut einen Punkt. Diesmal ist es allerdings wichtig, dass dieser an einer genauen Position ist. Setze ihn in die untere rechte Ecke des Bauteils genau 40mm von der unteren und 40mm von der rechten Kante entfernt. Teste dabei die "Einrasten"-einstellung siehe Menü.
4. Aufgabe	Lösche den eben erstellten Punkt.
5 . Aufgabe	Wir befinden uns nun wieder im Anfangsmenü. Dein neues Ziel ist es, die gesamte obere Kante des Bauteils (mit dem Finger zeigen) zu markieren. Nutze dafür das Abschnittemenü. Kleine Erinnerung: Auswählen bedeutet hierbei, dass du es als beispielsweise zu schweißen markierst. Da das Bauteil eben an dieser Kante an ein anderes zu schweißen ist in unserem Beispiel.
6. Aufgabe	Ändere nun die Richtung des Segmentes. Die Richtung gibt an, von welchem Punkt zu welchem anderen Punkt die Segmentsetzung vorgenommen wird.
7. Aufgabe	Die obere Kante ist nun ausgewählt. Jedoch ist uns aufgefallen, dass wir nun - statt nur der oberen Kante - die obere Kante und die rechte Kante zusammen auswählen wollen. Versuche die ausgewählte Strecke so zu verändern, dass das Segment die obere und rechte Kante abdeckt.
8. Aufgabe	"Setze das rechte Nahtende nun auf 40mm Abstand zur unteren Kante. Nutze hierbei verschiedene Einstellungen des Übersetzungsfaktors."
9. Aufgabe	"Ändere nun das Segment so, dass die Punkte auf anderem Wege miteinander verbunden werden. (Noch eine setzen, falls schon alles gelöscht wurde.)"
10. Aufgabe	Lösche alle erstellten Strecken. (Noch eine setzen, falls schon alles gelöscht wurde.)
Endbefragung	Im Folgenden werden Dir zum Abschluss weitere sechs Fragen gestellt, welche dir auf der gleichen Skala von 1 bis 5 wie zuvor beantworten kannst. Allerdings 1 bei den Fragen von 1-5 für sehr schlecht und 5 für sehr gut. 3 steht folglich für neutral. Nur Frage 6 ist zwischen 1 - "Nein, stimme gar nicht zu" und 5 - "Ja, ich stimme sehr zu" zu beurteilen.

Verabschiedung	Vielen Dank für Deine Teilnahme an unserer Befragung.

Tabelle 6.3: Endbefragung der Testung

6.7 Auswertung der Testung

Die Auswertung der erhobenen Daten wird sich auf die quantitativen Werte der Fragebögen und die Zitate beziehen. Dafür werden vorerst deskriptive statistische Werte herausgearbeitet, welche darauf hin auf die in Abschnitt 6.4 erläuterten Usabilityaspekte bezogen werden.

6.7.1 Auswertung Fragen der Aufgaben

		Gesamtstichprobe			Technisch/ Informatisch	Human Factors	Δ
Fragen		Mean	Min	Max	Mean	Mean	Differenz (TI-HF)
Frage 1	Hast Du die Interaktion als konsistent empfunden?	3.82	3.3	4.7	4.08	3.5	0.58
Frage 2	Hat das System Dir genügend informatives Feedback über deine Eingaben geliefert?	3.82	2.8	5	4.19	3.39	0.8
Frage 3	Hat das System Dir ausreichend Informationen über seinen Status geliefert?	3.66	2.8	4.7	3.93	3.39	0.54
Frage 4	Hast du das Gefühl gehabt, die Kontrolle über das System gehabt zu haben?	3.46	2.6	4.6	3.82	3.28	0.54
Frage 5	Hat die genutzte Funktion so funktioniert wie Du es erwartet hast?	3.62	2.8	4.3	3.85	3.84	0.01

Abbildung 6.4: Deskriptive Daten - Aufgabenbefragung

Wie sich in der Abbildung 6.4 zeigt, wurde im Mittel ein Wert von 3 bis 4 bzw. zwischen “ich weiß es nicht/neutral” bis “ja, stimme zu” vergeben. Eine Tendenz zur Mitte (3) wurde zuvor vermutet, da diese bei einer ungeraden Anzahl an Auswahloption häufig der Fall ist.

Bei der Betrachtung der Werte fällt auf, dass die Versuchspersonen der Gruppe “Technisch / Informatisch” höhere Werte in den Fragen 1 bis 4 abgegeben haben. Die graue Skalierung in der letzten Spalte gibt diese an, wobei vom der Wert der Gruppe “Technisch / Informatisch” der Wert der Gruppe “Human Factors” abgezogen wurde. Demnach scheint die erste Gruppe eher zu einem positiven Feedback zu neigen als die Human Factors Befragten. Zur besseren Darstellung dient Abbildung 6.5. Sichtbar wird zudem, dass die Gruppe “Technisch / Informatisch” eher die Fragen 1-3 positiv bewertete / zustimmte und die Gruppe “Human Factors” fast gegenteilig die Fragen 1-4 eher weniger positiv - also neutral - ansah.

Aufgrund der geringen Anzahlen werden an dieser Stelle keine inferenzstatistischen Berechnung zur Signifikanzfeststellung verwendet.

Wie die Abbildung 6.6 zeigt, steigt der Mittelwert über alle fünf Fragen und alle fünf Befragten von Aufgabe 1 zu 10. Zudem wird ersichtlich den Nutzern anscheinend die

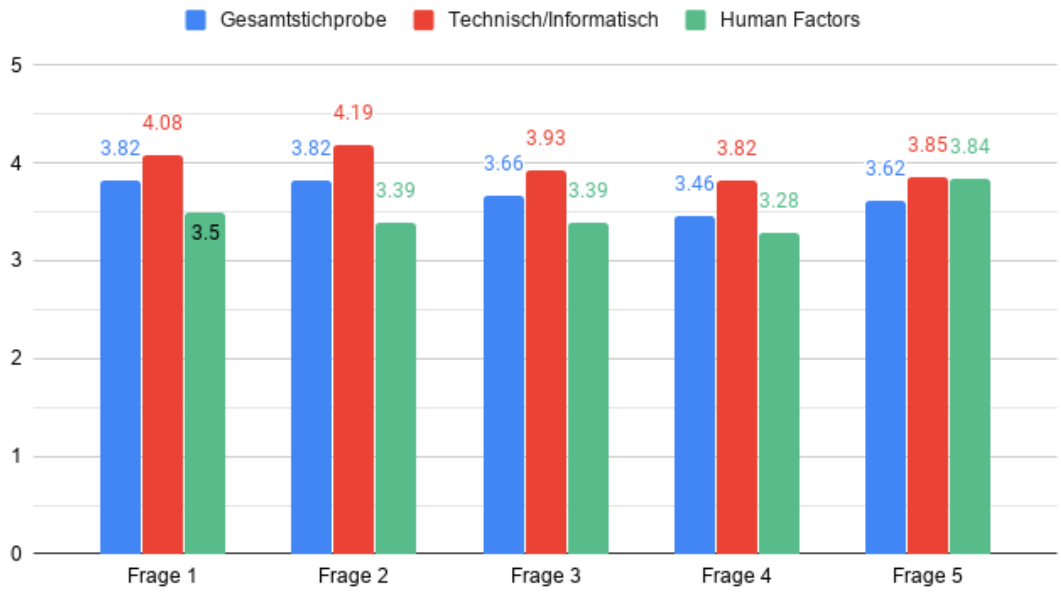

Abbildung 6.5: Deskriptive Daten - Aufgabenbefragung im Vergleich

Aufgaben	Art	Mittelwert über alle Fragen
1	Punkt setzen	2.36
2	Punkt löschen	3.36
3	Genauen Punkt setzen Einrastenfunktion	2.8
4	Punkt löschen	3.76
5	Strecke auswählen	3.36
6	Streckenrichtung ändern	4.44
7	Strecke erweitern	3.72
8	Übersetungsfaktor	4.24
9	Nicht-Strecke und Strecke umwandeln	4.04
10	Strecke löschen	4.68
Gesamt		3.68

Abbildung 6.6: Mittelwerte über alle fünf Fragen je Aufgabe

Aufgaben 6 - Streckenrichtung ändern und 10 - Strecke löschen am ehesten bezüglich der Konsistenz der Interaktion, dem informativen Feedback, der Statusanzeige, der Kontrolle und der Erwartungskonformität zusagten. Ebenso stellt sich der Übersetzungsfaktor (Aufgabe 8) und das Umwandeln der Strecken (9) als positiv heraus, da ein Mittelwert von über vier erreicht worden ist, welche für “ja, stimme zu” steht. Weniger gut bezüglich der eben genannten Kriterien scheinen die Aufgaben 1 - Punkt setzen und 3 - genauen Punkt setzen zu sein. Hierbei muss jedoch darauf hingewiesen, dass z.B. die Konsistenzfrage bei den Testpersonen zu Beginn nicht verständlich erschien, da noch keine Vergleichserfahrungen vorhanden waren.

6.7.2 Auswertung der Endbefragung

Die Endbefragung bestehend aus sechs abschließenden Fragen wurde ebenso deskriptiv statistisch betrachtet. In Abbildung 6.7 werden hierfür alle Mittel-, Minimal- und Maximalwerte je Frage über alle fünf Teilnehmer hinweg dargestellt.

Besonders zustimmend wurde die 3. Frage “Wie verständlich waren für dich die Buttonbezeichnungen? (1-5)” mit einem Mittelwert von 4.6 beantwortet. Da fünf die positivste Antwortmöglichkeit ist, kann somit vermutet werden, dass die Buttonbezeichnungen als sehr gut von den Testpersonen eingeschätzt wurden. Ebenso als gut wurden die 5. Frage “Wie gut fandest du die Menüführung? (1-5)” mit einem Mittelwert von 4.2 und die 1. “Frage Was hältst du von der Lage des Menüs?” mit einem Wert von 4 bewertet. Demnach Menüführung und die Lage dessen als positiv zu bewerten. Eher als neutral mit Tendenz zu gut wurden 2. Frage “Wie verständlich waren für dich die Überschriften der Teilfunktionen? (1-5)” und 4. Frage “Wie verständlich waren für dich die Anweisungen (oben Mitte)? (1-5)” beide mit einem Wert von 3.4 eingeschätzt. Demnach scheinen die Überschriften den Teilfunktionen und die Anweisungen als passend angenommen wurden zu sein. Ein etwas besserer Wert von 3.8 wurde für 6. Frage “Hast du die Implementierung der Funktionen zum Bearbeiten der Punkte und Abschnitte als konsistent empfunden?” abgegeben, welche hierbei dafür steht, dass die Testpersonen eher zustimmten, dass die eingebauten Funktionen als konsistent empfunden wurden.

	Endbefragung (Skala: 1 sehr schlecht - 5 sehr gut)	Usability	Mittelwerte	Min	Max
1. Frage	Was hältst du von der Lage des Menüs?	Erkennbarkeit, Lesbarkeit	4	3	5
2. Frage	Wie verständlich waren für dich die Überschriften der Teilfunktionen?	Lesbarkeit	3.4	3	4
3. Frage	Wie verständlich waren für dich die Buttonbezeichnungen?	Lesbarkeit	4.6	4	5
4. Frage	Wie verständlich waren für dich die Anweisungen (oben mitte)?	Lesbarkeit	3.4	3	4
5. Frage	Wie gut fandest du die Menüführung?	allgemeine Funktionalitäten	4.2	3	5
6. Frage	Hast du die Implementierung der Funktionen zum Bearbeiten der Punkte und Abschnitte als konsistent empfunden?	Konsistenz	3.8	3	5

Abbildung 6.7: Mittelwerte, Minimal- und Maximalwerte der Endbefragung aller sechs Fragen

6.7.3 Auswertung / Zusammenstellung der Zitate

Im folgenden werden die abgebenen Zitate von Interesse gelistet.

Testungsablauf	Zitate
Einführung	-
1. Aufgabe	• Zu Frage 1: “Es ist komisch, dass ich erst eine Kante auswählen muss, bevor ich einen Punkt auswählen kann.” • Zu Frage 5: “Ich dachte der Punkt muss auf der Kante liegen, weil ich nicht weiß, was mir ein Punkt in dem Objekt bringt. Als ich wusste, dass ich einen Punkt in dem Objekt legen kann, war mir unklar warum ich eine Kante wählen muss. Und es sollten wenn zwei gewählt werden.” und nach dem Erkennen der Bezugskante “Die Logik finde ich verwirrend.” - nach dem Erkennen der Bezugskante “Die Logik finde ich verwirrend.”
2. Aufgabe	-
3. Aufgabe	Allgemein: “Die Funktion von Einrasten ist unklar. Nach dem Testen war es klar.”
4. Aufgabe	-
5 . Aufgabe	Allgemein: “Keine direkte Information, dass dies tatsächlich eine Schweißnaht o.ä. ist.”
6. Aufgabe	-
7. Aufgabe	• Zu Frage 3: “Man sieht nicht, ob die ganze Kante markiert ist.” • Zu Frage 5: “Etwas stockend, schwer zu ziehen.”
8. Aufgabe	-
9. Aufgabe	-
10. Aufgabe	-
Endbefragung	• 1. Frage: “Für Linkshänder vielleicht Seiten tauschen.” • 5. Frage: “Abbrechbutton fehlt.”
Verabschiedung	

Tabelle 6.4: Endbefragung der Testung

Zusammenfassend wurden folgende Aspekte betont, Probleme herausgestellt und Verbesserungen vorgeschlagen:

- Unintuitive Kantenauswahl vor der Punktsetzung
- Weiß ist bei der Kantenauswahl nicht stark genug als Hervorhebungsfarbe
- Einrasten als Funktionsbeschreibung zuerst verwirrend, aber durch Testen verständlich
- Schweißnaht nicht eindeutig genug als solche dargestellt
- Gewählte Kante nicht gut erkennbar bei Streckenauswahl
- Endpunkte der Segmente schwer zu verschieben
- Linkshänder sollten Menülage tauschen können
- Abbrechbutton gewünscht

7 Anpassungsempfehlungen

Im Folgenden wird sich auf die eben genannten Ergebnisse bzw. die Auswertung der Usabilitytestung bezogen. Dabei werden die Ergebnisse nun mit den Usabilityaspekten, welche den gestellten Fragen zugrunde liegen, aus Kapitel 6.4 in Verbindung gebracht und diskutiert.

7.1 Ergebnisse der Aufgabenbefragung bezüglich Usabilityaspekten

Fragen	Bewertung	Usabilityaspekt und Anpassungsempfehlungen
1. "Hast Du die Interaktion als konsistent empfunden? "	3.82	*Konsistenz* Die wahrgenommene Konsistenz erscheint bereits ausreichend. Ein optimaler Wert könnte eventuell mittels eines Tutorials erreicht werden.
2. "Hat das System Dir genügend informatives Feedback über deine Eingaben geliefert?"	3.82	*Informatives Feedback, Steuerbarkeit* Das Feedback wurde positiv bewertet, die obige Range zeigt dennoch, dass dies nicht die Meinung aller Testenden war. Dabei könnte es helfen, z.B. die Anweisungen im obigen Sichtfeld zu verbessern oder eine Animation / Symbolanzeige einzubauen, welche Eingaben bestätigt. Auch eine Farbänderung könnte helfen, wie z.B. ein grünes Aufleuchten, wenn die Einrasteneinstellung eingestellt wurde.

3. "Hat das System Dir ausreichend Informationen über seinen Status geliefert?"	3.66	*Informatives Feedback, Benutzerkontrolle, Kurzzeitgedächtnis entlasten, Steuerbarkeit* Das Feedback wurde positiv bewertet, allerdings mit einer hohen Standardabweichung. Der aktuelle Status könnte demnach noch ausgeprägter dargestellt werden. Beispielsweise die in Frage 2. erwähnten Animationen könnten helfen. Zudem könnte es farblich markiert sein, wenn z.B. die Einrastenfunktion aktiv genutzt wird, damit dem Nutzer bewusst ist, dass diese in einer spezifischen Einstellung ist.
4. "Hast du das Gefühl gehabt, die Kontrolle über das System gehabt zu haben?"	3.46	*Benutzerkontrolle gewährleisten, Steuerbarkeit, Lesbarkeit, Unterscheidbarkeit, Erkennbarkeit* Auch hier ist die Standardabweichung hoch, was den hohen Mittelwert infrage stellt. Die gefühlte Benutzerkontrolle könnte erhöht werden, indem beispielsweise eine Zurück- oder Abbruchfunktion hinzugefügt wird. Dies wäre z.B. bei dem Zwischenschritt Kantenauswahl denkbar. Des Weiteren könnte die Software bezüglich des Benutzergefühles des Verschieben des Endpunktes eines Segmentes optimiert werden, da dies den Testpersonen schwer gefallen ist. Eine Funktion "Alles Löschen"könnte hierbei auch behilflich sein.

5. "Hat die genutzte Funktion so funktioniert wie Du es erwartet hast?"	3.62	*Erwartungskonformität* Auch hier ist die Standardabweichung groß genug, um Anpassungsideen als angemessen zu sehen. Besonders die Funktion des Punktesetzens erschien den Testenden unintuitiv. Folglich sollte hier an der Darstellung gearbeitet werden. Anstatt vor dem Setzen die Kantenauswahl vorzunehmen, könnte diese in dem Menü selbst stattfinden. Dabei könnte man einen Button mit der Funktion "Kante für Nullpunkt auswählen" zusätzlich ergänzen. Zudem sollten Buttons wie "Einrasten", welche nicht absolut intuitiv sind Funktionen wie Popuperklärungen haben oder diese sollten in einem Erklärunterpunkt oder Tutorial klargestellt werden.

Tabelle 7.1: Fragen bezüglich der Testaufgaben

7.2 Ergebnisse der Endbefragung bezüglich Usabilityaspekten

Fragen der Endbefragung	Bewertung	Usabilityaspekt
1. Frage "Was hältst du von der Lage des Menüs?"	4	*Erkennbarkeit, Lesbarkeit* Die Lage des Menüs erscheint passend. Dennoch gäbe es Verbesserungsmöglichkeiten wie beispielsweise eine Verschiebung der Lage. Je nach Bauobjekt könnte auch eine Leiste unten oder links nützlich sein.

2. Frage "Wie verständlich waren für dich die Überschriften der Teilfunktionen? (1-5)"	3.4	*Lesbarkeit* Die Überschriften wurden als gut bewertet mit Verbesserungsbedarf. Demnach sollte an der Verständlichkeit dieser gearbeitet werden. Die Überschriften "Punkte" und "Abschnitte" erschienen ausreichend selbsterklärend, da es keine Nachfragen gab. "Einrasten" und "Übersetzungsfaktor" hingegen waren nicht absolut klar. Hier könnte eine Tooltip-Informationsanzeige helfen, die bei längerem Hovern über dem Projekt angezeigt wird. Alternativ könnte eine Art Glossar angezeigt werden und ggf. ein Tutorial.
3. Frage "Wie verständlich waren für dich die Buttonbezeichnungen? (1-5)"	4.6	*Lesbarkeit* Es gab keine Nachfragen bezüglich dieser und der Score bestätigt, dass diese als sehr verständlich angesehen worden sind. "Richtung umkehren" beispielsweise wurde direkt korrekt angewandt und löste die vom Nutzer gewünschte Reaktion aus. Jedoch wurde die Darstellung der Pfeile etwas bemängelt. Eventuell könnte die Richtungsangabe verbessert werden, indem je Richtung eine andere Farbe genutzt werden würde. Auch könnte ein Pfeil außerhalb des Bauteiles zu Beginn des Segmentes die Richtungsanzeige übernehmen. "Abschnitt invertieren" wurde ebenso korrekt eingeschätzt. Dabei ist es günstig, dass zweimaliges Invertieren wie eine Zurück-Funktion agiert.

4. Frage "Wie verständlich waren für dich die Anweisungen (oben Mitte)? (1-5)"	3.4	*Lesbarkeit* Die Anweisungen selbst wurden teils als hilfreich beschrieben. Jedoch sollte die Formulierung dieser korrigiert bzw. verbessert werden. Es sollte sich auf eine Satz- oder Stichpunktstruktur geeinigt werden. Hier wäre auch die Option, die Erklärungen für die Einrastfunktion etc. abzugeben. Da die Tips teils nicht sehr auffielen bei zwei Testpersonen, könnte man überlegen, deren Farbe extremer zu gestalten oder Einblend- oder Aufleuchtanimationen einzuführen.
5. Frage "Wie gut fandest du die Menüführung? (1-5)"	4.2	*allgemeine Funktionalitäten* Die Menüfuhrung wurde gut bewertet. Dennoch wurde sich in den Zitaten dafür ausgesprochen, eine Abbruchfunktion einzubauen. Ebenso ist der Kantenauswahlstep als nicht intuitiv wahrgenommen wurden.

6. Frage "Hast du die Implementierung der Funktionen zum Bearbeiten der Punkte und Abschnitte als konsistent empfunden?"	3.8	*Konsistenz* Die allgemeine Funktionsimplementierung ist als gut einzustufen. Demnach können Details noch verbessert werden. Gut erschien auf jeden Fall die Farbwahl der Buttons "Löschen" und "Bestätigen", welche bei der Konsistenz halfen. Auch die Einrastfunktion wurde gleich dargestellt. Dabei könnte der Einrastschritt vor dem eigentlichen Punktmenü genauso wie die Kantenauswahl beim Abschnittsmenü weggelassen werden, da der Punkt sowieso noch verschiebbar ist und im eigentlichen Menü hierfür sowohl die Einrastfunktion als auch die Übersetzungsfaktorfunktion zur Verfügung stehen.

Tabelle 7.2: Endbefragung der Testung

7.3 Zusammenfassung der Zitate bezüglich Usabilityaspekten

- Unklare Kantenauswahl vor der Punktsetzung
- Abbrechbutton gewünscht
- Weiß ist bei der Kantenauswahl nicht stark genug als Hervorhebungsfarbe
- Einrasten als Funktionsbeschreibung zuerst verwirrend, aber durch Testen verständlich
- Gewählte Kante nicht gut erkennbar bei Kantenauswahl
- Endpunkte der Segmente schwer zu verschieben
- Linkshänder sollten Menülage tauschen können

7.4 Zusammenfassung der Anpassungsempfehlungen

Im Folgenden werden die obigen Empfehlungen in Kategorien zusammengefasst.

- Menüführung
  - Weglassen der Kantenauswahl (Abschnittsmenü) und Einrastfunktion/Kantenauswahl vor der Punktsetzung, da dies auch im Menü selbst eingebaut werden könnte (Button "Kante auswählen" einführen)

  - Kantenauswahl: Unklar was rotmarkierte macht, bzw. warum nur eine Bezugskante. Hier sollte es dem Nutzer intuitiver gemacht werden. Zb. Beim Punktsetzen, könnte die Kantenauswahl in den Anweisungen erst im Menü selbst erfolgen und erklärt werden.
  - Menülage variabel gestalten oder mindestens für Links- und Rechtshänder zwischen links und rechts einstellbar machen
- Anweisungen und Erklärungen
  - Tutorial, welches konsistentes Design von Beginn an klarstellt ODER Glossar für Definitionen und Anweisungen
  - Pop-Up-Informationen, wenn auf Bezeichnungen wie Einrasten gedrückt wird mit Definition/Erklärung
  - Anweisungen in Feedback einbeziehen / spezifischere Anweisungen wenn z.B. Punkt gesetzt ist, wie dieser wieder auszuwählen ist oder wie diese zu löschen/bestätigen ist
  - Anweisungen verdeutlichen über Animation/Einblendungen, um Status zu betonen
  - Anweisungen einheitlich als Stichpunkte/Befehle formulieren
  - neue Anweisungen aufleuchten lassen/farblich hervorheben
- Funktionen
  - Zurückfunktion einführen (optional, da es mit "Löschen" auch als gut nutzbar bewertet wurde)
  - Funktion "Alles Löschen" ergänzen
  - Bei Nutzen von Einrasten/Übersetzen (wenn nicht auf 1:1 bzw x1) dieses farblich hervorheben
  - Abbruchfunktion in Vormenü der Punkt- und Streckensetzung bzw. allgemeine Abbruchfunktion, die zu Hauptmenü zurück führt
  - "Einrasten" könnte mit einem anderen Wort betitelt werden wie z.B. Verschiebungseinheitsgröße. Da auch solche Namen nicht selbsterklärend sind, empfiehlt sich die obige Lösung: das Einfügen einer Erklärung.
- Richtungsanzeige: Verschiedene Richtungspfeile in verschiedenen Farben darstellen. Statt Richtungsanzeigen auf der Linie, könnte dies auch außerhalb des Objektes zu Beginn der Naht als Pfeil außerhalb sein. Dabei sollte der Abschnitt und der Pfeil je Richtung eine eigene Farbe haben.
- Streckenziehen könnte intern so programmiert werden, dass es dem Nutzer leichter fällt, diese zu ziehen

Zusammenfassend ist zu erwähnen, dass das Gesamtfazit der Befragten positiv ausfiel. Diese betonten, dass die bisherige Anwendung bereits benutzbar, vor allem nach einer kurzen Selbsterkundungsphase. Folglich sind die obigen Anpassungsempfehlungen

auf eine Usabilityverbesserung bzw. User-Experience-Verbesserung bezogen und keine Notwendigkeit, um die Anwendung sinnvoll verwenden zu können. Dennoch sollte die Mehrheit der genannten Punkte beachtet und bearbeitet werden, da gerade bei vermehrter Verwendung eine suboptimale Menüführung oder unnötige Zwischenschritte die erforderliche Zeit erhöhen und die Nutzerzufriedenheit senken.

8 Fazit

8.1 Reflexion

Wie in jedem Projekt gibt es Aspekte die rückblickend besser oder zumindest anders gehandhabt hätten werden können, oder bei denen die praktische Ausführung nicht der eigentlich Planung entsprochen hat. Diese Aspekte betrafen vor allem die Ablaufsplanung und Zielsetzung des Projektes. Die größten Reibungspunkte und Hürden, die es zu überwinden galt, traten in folgenden Bereichen auf:

- Die Anforderungsanalyse gestaltete sich schwer, da Rahmenbedingungen und Anwendungsfall nicht fest definiert waren
- Verschiedene Kenntnisbereiche der Gruppenmitglieder machten interdisziplinäres Arbeiten erforderlich. Diese musste vor allem bei der Aufgabenaufteilung und der zeitlichen Abfolge der Aufgaben mit einbezogen werden, welche strukturierter und disziplinierter ablaufen können.
- Ein wichtiger Aspekt hierbei war auch, dass die meisten Gruppenmitglieder gleichzeitig weitere arbeitsintensive Module belegten, welches aber im initialen Zeitplan nicht mit einbezogen wurde. Hier hätte ggf. offenere Kommunikation hilfreich sein können.
- Auch die zuerst vorgesehene struktuierte agile Projektplanung wurde deshalb nicht eingehalten, insbesondere das Timeboxing
- Zudem wurde der Stand des bestehenden Prototyps falsch eingeschätzt: Dieser war leider nicht als Grundlage geeignet, auch da dieser kaum dokumentiert war. Das vorgenommene Ziel von 2 zu vergleichenden Versionen war somit nicht erreichbar.
- In Folge dessen musste die Usability-Testung entsprechend umgestellt werden und konnte erst spät im Projektverlauf erfolgen.
- Für die Kommunikationssprache haben wir uns als Gruppe auf Deutsch geeinigt. U.a. aufgrund der vielen technischen Begriffe war dies sprachlich herausfordernd; ggf. wäre Englisch für die zwei internationalen Gruppenmitglieder einfacher gewesen.
- Eine weitere Problematik war das nicht kommunizierte Austreten von einem der fünf Gruppenmitgliedern etwa zur Hälfte des Projektes. Zwar konnte auch ohne dieses Mitglied das Projekt abgeschlossen werden; dennoch ist zu erwähnen, dass das angesetzte Ziel ein Arbeitspensum für fünf Personen umfasste.

Jedoch sind auch einige Aspekte positiv hervorzuheben. So hat sich die Gruppenkommunikation mittels einer Telegram-Gruppe als sehr effizient herausgestellt, wodurch nur wenige Teamtreffen erforderlich waren. Auch der Umgang und die Absprachen in der Gruppe verliefen unkompliziert und professionell: Fragen wurden zügig beantwortet, Termine schnell gefunden und Vereinbarungen so gut es ging eingehalten. Darüber hinaus wurde auch auf persönliche Umstände (z.B. in Bezug auf die Zeitplanung des Projektes) respektvoll reagiert und Rücksicht genommen. Letztendlich verlief auch die Kommunikation mit den Projektbetreuenden unkompliziert und effektiv.

8.2 Ausblick

Das Projekt ist sowohl in seiner Konzeption als auch in seinem Resultat als Fundament und Gerüst für weitere spezifische Features der AR-Anwendungen zu sehen. Unter diesem Aspekt sehen wir für die nähere Zukunft des Projektes vor allem eine kontinuierliche Weiterentwicklung. Hierbei konnte vor allem die Usability-Testung aufzeigen, welche Aspekte bereits zufriedenstellend sind und in welchen Bereichen noch Verbesserungsbedarf besteht. Die Usabilitytestung selbst konnte mittels theoretischem Usabilitywissen erstellt werden und ist nach geplanten Standards umgesetzt wurden. Dennoch könnte es sich lohnen, hierfür zusätzlich standardisierte Skalen wie die erwähnte Usability-Scale SUS zu verwenden. Auch könnten es hilfreich sein, Lautes Denken als Methodik zu nutzen, um noch spezifischere Anpassungsempfehlungen zu erhalten. Des Weiteren wäre es ratsam, eine ähnliche Testung in der eigentlichen AR-Projektor-Umgebung vorzunehmen. Dies würde zum einen etwaige unentdeckt gebliebene Problemstellen aufdecken und zum anderen eine zusätzliche Untersuchung darstellen, welche aufzeigt, ob Touchumgebungen repräsentativ für AR-Projektor-Umgebungen bezüglich ihrer Usability und Funktionalitätsdarstellung sind. Darüber hinaus steht vor allem die Integration mit der Robotersteuerung aus. Zudem wird vermutlich auch die Eingabe von Prozessparametern, wie z.B. Schweißnahtdicke oder Bohrlochtiefe ein zukünftiger Fokusbereich sein. Eine detaillierte Auflistung der Verbesserungsvorschläge auf Grundlage der Usability-Testung befindet sich in 7, technische Anspassungsvorschläge und -anleitungen befinden sich in 5.

9 Anhang

Zusätzlich zu diesem Abschlussbericht besteht die Abgabe aus den folgenden digitalen Anhängen:

- **BuildWindows.zip:** Eine kompilierte Testversion der entwickelten Anwendung
- **Dokumentation_HTML.zip:** Die mittels Doxygen generierte API-Dokumentation
- **projectorPlacement.zip:** Das Unity-Projekt, inkl. Quellcode

Literatur

Akkreditierungsstelle, Deutsche (2010). „Leitfaden Usability“. In: *Gestaltungsrahmen für den Usability-Engineering-Prozess. Hg. v. Deutsche Akkreditierungsstelle*.

Brooke, John et al. (1996). „SUS-A quick and dirty usability scale“. In: *Usability evaluation in industry* 189.194, S. 4–7.

Cabral, Marcio C, Carlos H Morimoto und Marcelo K Zuffo (2005). „On the usability of gesture interfaces in virtual reality environments“. In: *Proceedings of the 2005 Latin American conference on Human-computer interaction*. ACM, S. 100–108.

Dix, Alan et al. (2003). *Human-computer interaction*. Pearson Education.

Farhadi-Niaki, Farzin, S Ali Etemad und Ali Arya (2013). „Design and usability analysis of gesture-based control for common desktop tasks“. In: *International Conference on Human-Computer Interaction*. Springer, S. 215–224.

Forward, Andrew und Timothy C. Lethbridge (2002). „The Relevance of Software Documentation, Tools and Technologies: A Survey“. In: *Proceedings of the 2002 ACM Symposium on Document Engineering*. DocEng '02. McLean, Virginia, USA: Association for Computing Machinery, S. 26–33. ISBN: 1581135947. DOI: 10.1145/585058.585065. URL: https://doi.org/10.1145/585058.585065.

Gamma, Erich et al. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201633612.

Hornbogen (2018). *Wie die 7 Dialogprinzipien Usability Engineers beim Design helfen*. URL: https://blogs.itemis.com/de/wie-die-7-dialogprinzipien-usability-engineers-beim-design-helfen (besucht am 20. 03. 2020).

Kuss, Alexander et al. (2017). „Manufacturing task description for robotic welding and automatic feature recognition on product CAD models“. In: *Procedia CIRP* 60, S. 122–127.

Mazumder, Fourcan Karim und Utpal Kanti Das (2014). „Usability guidelines for usable user interface“. In: *International Journal of Research in Engineering and Technology* 3.9, S. 79–82.

Meng, Michael, Stephanie Steinhardt und Andreas Schubert (2018). „Application Programming Interface Documentation: What Do Software Developers Want?“ In: *Journal of Technical Writing and Communication* 48.3, S. 295–330. DOI: 10.1177/0047281617721853. URL: https://doi.org/10.1177/0047281617721853.

Paelke, Volker et al. (2015). „User interfaces for cyber-physical systems“. In: *at-Automatisierungstechnik* 63.10, S. 833–843.

Rossano, Gregory F et al. (2013). „Easy robot programming concepts: An industrial perspective“. In: *2013 IEEE international conference on automation science and engineering (CASE)*. IEEE, S. 1119–1126.