Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Ricardo Möhler

Erstellung einer Augmented Reality-Testumgebung mit Matlab

Ricardo Möhler

Erstellung einer Augmented Reality-Testumgebung mit Matlab

Fakultät Technik und Informatik
Department Fahrzeugtechnik und Flugzeugbau

Faculty of Engineering and Computer Science
Department of Automotive and Aeronautical Engineering

Studienarbeit eingereicht im Rahmen der Prüfungsleistung: Studienarbeit

im Studiengang Fahrzeugbau
am Department Fahrzeugtechnik und Flugzeugbau
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Erstprüfer: Prof. Dr. - Ing. Dirk Adamksi

Abgabedatum: 08.07.2021

Zusammenfassung
Ricardo Möhler

Thema der Bachelorarbeit

Erstellung einer Augmented Reality-Testumgebung mit Matlab

Kurzzusammenfassung

In dieser Arbeit wird untersucht und dargestellt wie eine „Augmented Reality“ Umgebung, beginnend mit einfachen Beispielen und Grundlagen, für die Darstellung von virtuellen Objekten entwickelt werden kann. Darüber hinaus soll die Darstellung von CAD Daten, in virtueller Form, durch optische Überlagerung, erzielt werden. Als Entwicklungsumgebung dient hier die Software „Matlab“ der Firma „The MathWorks, Inc.“, die mittels erwerblicher Add-Ons, den sogenannten „Toolboxes“ (anwendungsorientierten Werkzeugkisten), einen Zugang zu verschiedensten gängigen Computer-Vision Algorithmen gewährt. Die dazu aufrufbaren Funktionen werden getestet und bewertet, um sie anschließend in einen Algorithmus zu implementieren, der durch ein eigenständiges Skript realisiert wurde und die AR-Anwendung darstellt.

Subject of the thesis

Development of an augmented reality test environment with Matlab

Brief summary

In this thesis it will be examined and shown how an "Augmented Reality" environment, starting with simple examples and basics, can be developed for the representation of virtual objects. In addition, the representation of CAD data in virtual form, through optical overlay, is to be achieved. As development environment serves here the software "Matlab" of the company "The MathWorks, Inc.", which grants an access to most different usual computer vision algorithms by means of acquirable Add-Ons, the so-called "Toolboxes" (application-oriented tool boxes). The callable functions are tested and evaluated to implement them in an algorithm, which is realized by an independent script and represents the AR application.

Inhaltsverzeichnis

1	**Einleitung**	**6**
1.1	Kurzportrait: Augmented Reality	6
1.2	Motivation	6
1.3	Aufgabenstellung	7
1.4	Gliederung der Arbeit	7
2	**Grundlagen zur Umsetzung und Erstellung des Lösungskonzepts**	**8**
2.1	Grundlegende Funktionsweise von visueller AR	8
2.2	Notwendige Bestandteile und Algorithmen	9
2.3	Lastenheft mit Zielsetzung für eine prototypische Lösung in Matlab	14
2.4	Abgleich der Anforderungen mit verfügbaren Algorithmen u. Lösungen	16
2.5	Grundlegende Funktionsweisen v. essenziellen Algorithmen	20
2.5.1	Objekterkennungsalgorithmen	20
2.5.1.1	Testreihe z. Bewertung u. Auswahl e. geeigneten Objekterkennungsalg.	20
2.5.1.2	Funktionsweise u. Vorstellung des gewählten Objekterkennungsalg.	24
2.5.2	Objekttrackingalgorithmus	28
2.5.2.1	Funktionsweise und Vorstellung des gewählten Objekttrackingalg.	30
2.6	Synthese und Analyse von Lösungsmöglichkeiten für eine AR Umgebung	32
2.6.1	Lösungsideen und ihre Eigenschaften	33
2.7	Bewertung und Konzeptfestlegung	36
3	**Umsetzung und Vorstellung der Augmented-Reality Umgebung in Matlab**	**40**
3.1	Vorbereitungen und Annahmen für die Umsetzung auf einem mob. Endgerät	40
3.2	Erstell. einer einfachen AR Umgebung für die opt. Überlagerung v. 2D Obj.	40
3.2.1	Initialisierung der Randbedingungen und Objekte	40
3.2.2	Erste Schritte zur Realisierung eines Mappings zwischen einem realen Marker und dessen Referenz	41
3.2.3	Initiale Position: 2D	45
3.2.4	Darstellung/Optische Überlagerung: 2D	47
3.2.5	Tracking: 2D	48
3.2.6	Berechnung der neuen Positionierung und Darstellung: 2D	49
3.3	Erstellung einer einfachen Augmented Reality Umgebung für die optische Überlagerung von 3D Objekten/CAD Daten	51
3.3.1	Übernahme von entwickelten Ansätzen des 2D Beispiels	51
3.3.2	Schritte bei der Bearbeitung der 3D Augmented Reality Umgebung	51
3.3.3	Zwischenergebnisse und Optimierungsschleife	60
3.4	Implementierung der Endversion auf einem handelsüblichen Smartphone	64
4	**Zusammenfassung und Ausblick**	**66**
4.1	Abgleich mit der Zielsetzung und Einschränkungen	66
4.2	Lösungsansatz für eine Weiterentwicklung	68
5	**Quellenverweise**	**69**
Anhang A: Testreihe zu Objekterkennungsalgorithmen in Matlab		**72**
Anhang B: Augmented Reality Umgebung für die optische Überlagerung von 2D-Daten		**78**
Anhang C: Augmented Reality Umgebung für die optische Überlagerung von CAD-Daten		**84**
Anhang D: Kurzreferat		**94**
Eidesstattliche Erklärung		**97**

1 Einleitung

1.1 Kurzportrait: Augmented Reality

„Augmented Reality“ (kurz: “AR“), zu Deutsch: Erweiterte Realität, bezeichnet die interaktive Darstellung der Realität, die um computergenerierte, jedoch menschlich wahrnehmbare Daten erweitert wird. Teilweise werden mehrere Sinneseindrücke, wie optische und akustische Wahrnehmung kombiniert. Die genannten Eigenschaften für die Definition von AR, folgen denen von R. Azuma [1].


Abbildung 1 [41]

Die vorliegende Arbeit setzt sich jedoch mit einer speziellen Ausprägung von AR, der meist verbreiteten und bekanntesten Form, der **visuellen Erweiterung der Realität** auseinander. Dies geschieht durch das Überlagern von virtuellen Objekten in einem dreidimensionalen Zusammenhang mit der realen Umgebung. Bei dieser Form von AR werden 3D-Modelldaten oder 2D-Bilddaten genutzt und als virtuelles Objekt verstanden. Frei erwerbliche Smartphone Applikationen bieten zum Beispiel die Möglichkeit ein existierendes Fahrzeug in der realen Umgebung als virtuelle Erweiterung zu visualisieren.

1.2 Motivation

Die Anzahl der Anwendungsmöglichkeiten von AR ist lediglich durch den Innovationstrieb limitiert. Heutzutage gibt es zunehmend in jeder Branche eine Bereicherung durch erweiterte Realität.

Ein namhaftes Beispiel ist die Verwendung von Augmented-Reality in der Produktentwicklung der Automobilindustrie und als Produktbestandteil von Automobilen. Als Bereicherung für die Produktentwicklung nutzt man AR-Technologien bereits in der Design- und Konzeptphase des Produktentstehungsprozesses. Die Visualisierung von Formen und Designs ermöglichen eine virtuelle Betrachtung eines 3D Modells. Der Vorteil besteht darin, dass Zeit und Kosten aufgrund entfallender physischer Limitierungen gespart werden. Gleiches gilt für die Konstruktion, bei der man noch vor einer Prototypenfertigung, die Derivate eines Fahrzeugs oder einer Komponente kostengünstig, durch eine AR-CAD Darstellung visualisieren und gegeben falls darauf basierend Entscheidungen treffen kann.

In dieser Arbeit findet daher eine Fokussierung auf eine optische Erweiterung der Umgebung statt. Im Folgenden wird der Begriff optische basiertes AR System eingesetzt und mit den genannten Eigenschaften in Verbindung gebracht.

In der Produktion können unter anderem auch Mensch-Maschine Interaktionen mithilfe von virtuell angedeuteten Produktionsoperationen optimiert werden. Darüber hinaus kann AR auch selber eine Komponente des Produktes sein. Heutige Navigationssystem in Fahrzeugen verwenden beispielsweise eine optische Überlagerung von Symbolen und Beschriftungen mit der realen Umgebung des Fahrzeugs, um das Navigieren intuitiver zu gestalten. Im Allgemeinen besteht die Hautpmotivation, für die Entwicklung von AR Lösungen, darin die menschlichen Sinneswahrnehmungen derart anzusprechen, dass Prozessschritte oder die Wirkung von Zusammenhängen besser abstrahiert werden können.

1.3 Aufgabenstellung

lm Rahmen dieser Arbeit sollen die Möglichkeiten der Simulationsumgebung Matlab/Simulink genutzt werden, um eine einfache AR-Umgebung zu schaffen, die auf einer handelsüblichen Hardware (Smart Phone, Tablett) verwendet werden kann. Hierfür sind zunächst die notwendigen Bestandteile einer solchen Umgebung zu analysieren und mit den zur Verfügung stehenden Toolboxen von Matlab/Simulink abzugleichen. Die Objekterkennung und das Tracking sind zunächst an einfachen Beispielen umzusetzen. Nachdem die grundlegende Funktionsweise nachgewiesen wurde, soll untersucht werden, inwiefern vorhandene CAD-Daten eines realen Bauteils in der zu schaffenden Umgebung optisch überlagert werden können. Die verschiedenen Techniken z. B. zur Objekterkennung sind zu testen und zu bewerten. Die notwendigen Schritte zur Erstellung einer AR-Umgebung sind zu dokumentieren. Für die AR-Umgebung soll eine prototypische Anwendung programmiert und getestet werden.

1.4 Gliederung der Arbeit

Zu Beginn dieser Arbeit werden die Grundlagen zur Realisierung AR Umgebung gesammelt und analysiert. Nach der Klärung des Bedarfs werden die Bestandteile mit bestehenden Lösungen und verfügbaren Funktionen der anwendungsorientierten Bibliothek von Matlab abgeglichen und geeignete, zur Auswahl stehende, Funktionen ermittelt. Darauf basierend werden, als Grundlagen-Unterkapitel, die Funktionsweisen der essenziellsten Algorithmen näher untersucht und erläutert. Die quantifizierbaren Ziele und Anforderungen werden basierend auf dem zur Verfügung stehenden Entwicklungsrahmen in Form eines Lastenhefts dargestellt und ebenso der angestrebte Endzustand formuliert. Mit den feststehenden Zielen werden verschiedene Lösungsideen entwickelt, gesammelt und deren Eigenschaften analysiert. Die erarbeiteten Lösungsmöglichkeiten werden bewertet und im folgenden Unterkapitel ein daraus resultierendes, weiter zu verfolgendes, konkretes Lösungskonzept präzessiert.
Im dritten Kapitel wird die Ausarbeitung des entwickelten Lösungsansatzes dokumentiert und erläutert. Zunächst werden die notwendigen Vorarbeiten beschrieben und notwendige Annahmen

für die Umsetzung getroffen, die sich auf die konkrete praktische Programmierung des zu entwickelnden Skripts und die Entwicklungsumgebung beziehen. Im weiteren Verlauf von Kapitel drei, lässt sich dieses in zwei wesentliche Abschnitte unterteilen. Diese stellen zum einen die Umsetzung mit einer 2D Bild Darstellung als erste Entwicklungsstufe dar und zum anderen die Endstufe mit der Darstellung eines 3D CAD Models.
Im ersteren der beiden Abschnitte, 3.2, werden schrittweise die Vorgehensweise und die jeweiligen Meilensteine bei der Entwicklung erläutert.
In Abschnitt 3.3 findet zu Beginn eine kurze Reflexion und Übernahme der erreichten Ziele aus 3.2 statt.
Basierend auf dem zuvor erreichten Stand, wird die Entwicklung mit dem Ziel des Erreichens des gewünschten Endzustands, fortgesetzt. Auf diesem Weg werden die entstehenden Herausforderungen dargelegt und die erforderlichen Schritte erarbeitet. Zum Ende von Abschnitt 3.3 werden die erzielten Zwischenergebnisse und gleich darauf die angewandten Optimierungen erörtert, die letztlich zur Endversion der Entwicklung führen.
Das Kapitel schließt mit der Implementierung und den hierfür notwendigen Anpassungen auf ein mobiles Endgerät (Smartphone) und der Vorstellung des erreichten Endzustands des Prototyps und des Gesamtergebnisses dieser Arbeit ab.

Im letzten Kapitel werden die erreichten Ziele und inwieweit der erzielte Stand der Zielsetzung gerecht wird, behandelt. Des Weiteren werden auffällige Einschränkungen und auftretende offenen Fragen genannt. Daraus resultierend, werden offene Punkte diskutiert und Ideen für das nachfolgende theoretische Vorgehen unter Idealvorstellungen bestimmt.

2 Grundlagen zur Umsetzung und Erstellung des Lösungskonzepts

2.1 Grundlegende Funktionsweise von visueller AR

Unter Berücksichtigung der Begrifflichkeit und Eigenschaften für eine optisch basiertes Augmented Reality System gilt es die Funktionsweise abstrakt zu benennen und näher zu spezifizieren:
Die grundlegende Methode von AR Systemen lässt sich in fünf wesentliche Prozesschritte überordnen. Diese sind Bildaufnahme, Erkennung (Computer Vision), Tracking, Positionierung und Darstellung (Rendering) (s. Abb. 2). Im Folgenden werden die Schritte nähergebracht.
Im ersten Schritt, der Bildaufnahme, wird ein Abbild, ein statisches Bild, der vorliegenden realen Umgebung aufgezeichnet. Dieser Schritt geschieht mithilfe einer kalibrierten Kamera. Das Bild wird ständig durch eine Iteration neu aufgezeichnet, sodass ein bewegtes Bild aus der fortlaufenden Iteration hervorgeht. Zu jeder Iteration muss das Bild zuverlässig an den nächsten Prozessschritt übergeben werden. Diese wird nachfolgend beschrieben.

Als nächstes werden die Merkmale der Umgebung, innerhalb der Aufnahme, ermittelt. Die Erkennung von Merkmalen kann auf mehrere Arten erfolgen. Grundsätzlich kann man an dieser Stelle zwischen zwei Kategorien unterscheiden.

Demnach gibt es, zum einen, sogenannte **markerlose** AR Systeme, die aufgrund der Bildmerkmale der gesamten Umgebung eine Positionsbestimmung der Kamera in der realen Welt ermöglichen und zum anderen **markerbasierte** AR Systeme, die eine in der realen Welt befindliche eindeutige Markierung verwenden, um die initiale Position zu bestimmen (Für eine AR Wirkung müsste theoretisch die Position in Abhängigkeit des Betrachters bestimmt werden, da die Anwendungen jedoch häufig auf einem mobilen Endgerät umgesetzt sind, entspricht die Kameraposition annähernd der des Betrachters).

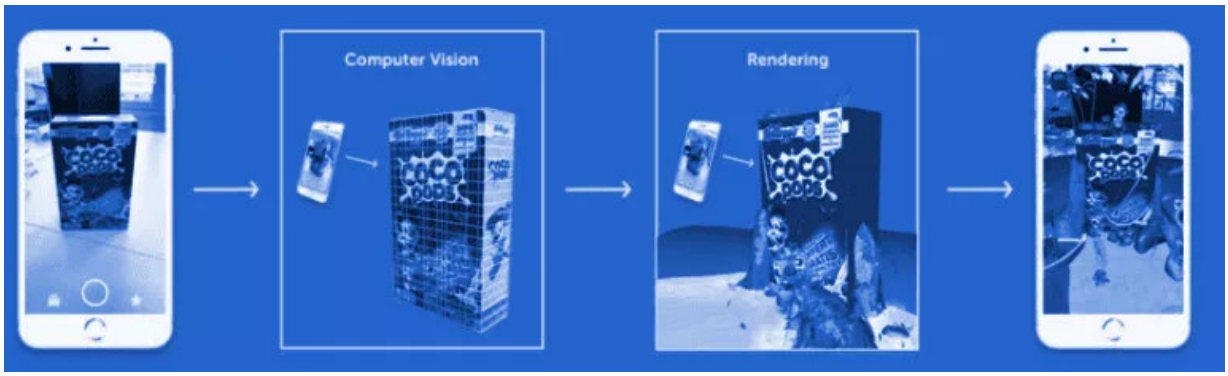


Abbildung 2 [2]

Darauffolgend findet ein Tracking der erkannten Merkmale statt. Unter dem Tracking versteht man das Verfolgen von bereits erkannten Bildpunkten, aus denen sich Positionen und Positionsänderungen ableiten lassen. Infolgedessen wird, basierend auf den Trackingpunkten, eine fixierte Position des einzusetzenden virtuellen Objekts, in der realen Welt der Aufnahme, in einem fiktiven Koordinaten System ermittelt.
Die Darstellung sorgt für die menschliche wahrnehmbare, erweiterte Realität. Das Objekts wird an der entsprechenden Stelle des zweidimensionalen Bild-Koordinatensystems substituiert. Hiernach lassen sich auch grundlegende Voraussetzungen für ein AR Umgebung beschreiben: Für die Bildaufnahme wird eine Kamera vorausgesetzt, die sich in unmittelbarer Nähe des Betrachters befindet. Außerdem muss aus den aufgezeichneten Bildern eine als euklidischer Vektorraum interpretierbare Umgebung, die sich durch eindeutige Koordinaten darstellen lässt, hervorgehen. Dies bedeutet, dass man dem Bild unserer optischen Anschauung nach als dreidimensionalen reellen Koordinatenraum betrachten kann. Ebenso gelten dieselben Bedingungen für das Objekt mit der eine optische Überlagerung erzielt werden soll. Diese Voraussetzungen gelten konkret in dem Fall einer visuellen erweiterten Realität, mit der sich diese Arbeit beschäftigt.

2.2 Notwendige Bestandteile und Algorithmen

Bezugnehmend auf die dargelegten Grundlagen einer visuell erweiterten Realität, sollen hiernach die notwendigen Bestandteile und Algorithmen durch das Analysieren der allgemeinen Funktionsweise ermittelt werden. Somit wird in diesem Abschnitt jeder Schritt aus 2.1 behandelt. Jedoch findet erst in 2.4 und 2.5 eine individuelle Fokussierung auf die konkreten Lösungen unterschiedlicher spezieller Algorithmen statt.

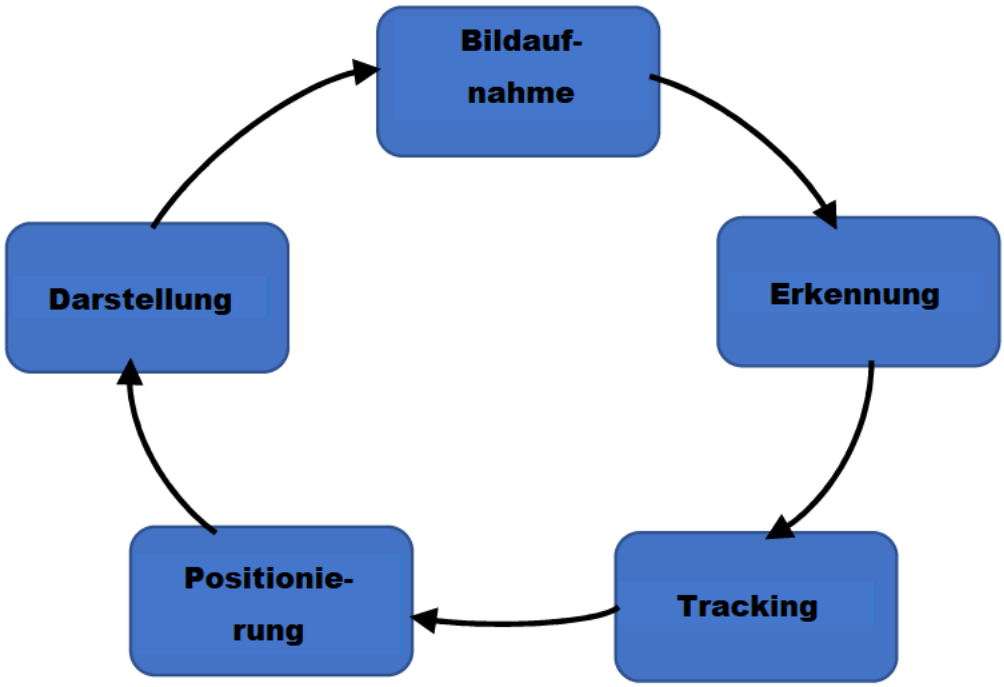


Abbildung 3

Bildaufnahme:

Für diesen Teil wird, wie bereits in 2.1 angedeutet eine kalibrierte Kamera benötigt.

Zunächst kann man festhalten, dass eine zeitgemäße handelsübliche Kamera notwendig ist, die gewisse Mindestanforderungen erfüllen sollte. Nach Beschreibungen von AR Anwendungen und Standards für zum Beispiel Smartphones, sollte die Kamera eine Mindestauflösung von 720p und eine Bildrate von mindestens 60fps bieten. Diese Werte gelten als etabliertes Minimum für ein ausreichendes AR Ergebnis [3].

Betrachtet man die Kalibrierung ist hiermit gemeint, dass die Einstellungen der Kamera, die sich softwareseitig einstellen lassen, auf den Anwendungsfall eines Augmented Reality Systems abgestimmt sein muss. Die Einstellungen sollen zu Beginn der Laufzeit der Anwendung, zum Beispiel als Bestandteil der Initialisierung geschehen. Dies bedeutet im Detail, dass die Kameraauflösung idealerweise auf die maximale verfügbare Auflösung gesetzt wird. Die nächste wichtige Einstellung stellt die Bildrate dar. An dieser Stelle sollte ebenfalls die höchstmögliche Einstellung an Bildern, die pro Sekunde aufgezeichnet werden können, angestrebt werden. Eine unzureichende Framerate pro Sekunde könnte einen Engpass für kürzere Laufzeiten darstellen. Hintergrund sind die ständig neu zu berechnenden Trackingpunkte, welche zwangsläufig auf der nächsten Bildaufnahme beruhen. Wird das nächstentstehende Bild bei einer Bewegung später

aufgenommen als die Berechnung des Trackings durchläuft, kommt es aufgrund der langsameren Komponente, der Bildrate, auch zu einer Verzögerung beim Tracking. Neben einer nicht für den Menschen stimmigen bewegten Aufnahme entstehen fehlende Trackingposition, die zu einem Abriss des Trackings führen können.
Des Weiteren sollten weitere Eigenschaften, wie die Bildschärfe, der Kontrast und die Helligkeit nach Wirkung auf die Trackingqualität angepasst. Infolgedessen wird eine exakte Kalibrierung und die nötigen Einstellungen erst mit der praktischen Anwendung der bereits existierenden Trackingroutinen möglich. Demnach müssen die genannten Anpassungen für eine Kalibrierung der Kamera und zur späteren Optimierung der Anwendung, parametrisch geändert werden können.

Objekt-/Merkmalserkennung:
Bei dem zweiten Schritt, der Merkmalserkennung, handelt es sich um einen Algorithmus, der in verschiedensten Prinzipien umgesetzt werden kann.
Beginnend mit der Frage, welchen Input dieser Schritt benötigt, lässt sich sagen, dass die Funktion Daten eines aufgezeichneten Bildes aus zu vorigem Schritt erwartet. Daher wird das Lesen von Bilddaten, vordefinierten Typs, eingesetzt.
Für die darauffolgende Frage, wie in der Verarbeitung die Merkmale erkannt werden, muss das Bild interpretiert und die Merkmale durch einen geeigneten Algorithmus bestimmt werden.
Als Merkmale werden in diesem Zusammenhang charakteristische Bildpunkte eines Bereiches verstanden, die auf binärem Niveau in dem zugrundeliegenden Bild ermittelt werden. Die Methode des Erkennens dieser Merkmale ist ein Teilgebiet der Computer Vision, die wiederum eine Überschneidung der Informatik und Ingenieurswissenschaften darstellt. Mit verschiedensten Methoden der Merkmalserkennung werden Abstraktionen von Bildinformationen berechnet und zur lokalen Entscheidung an jedem Bildpunkt, ob an diesem Punkt ein Bildmerkmal eines bestimmten Typs vorhanden ist oder nicht, genutzt. Die Aufgabe des Merkmalserkennungsalgorithmus, lässt sich mit folgendem Modell vereinfachen, in Abb. 4:

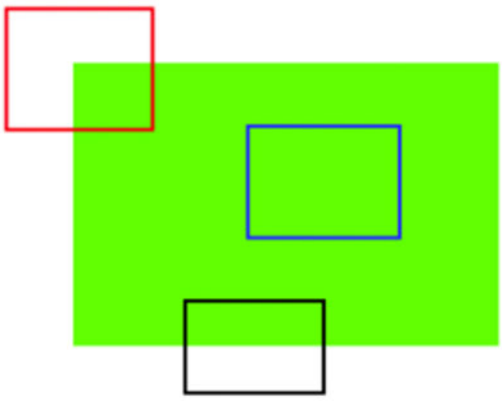


Abbildung 4 [4]

Das blaue Rechteck ist ein uncharakteristischer Bereich und schwer zu finden und zu verfolgen. Egal, wohin man den blauen Fleck bewegen wird, er sieht immer gleich aus.
Das schwarze Rechteck schließt eine Kante ein. Wenn man dieses in vertikaler Richtung (d. h. entlang des Gradienten) verschiebt, ändert er sich. Bewegt man ihn entlang der Kante (parallel

zur Kante), sieht er gleich aus. Das rote Rechteck jedoch, markiert eine Ecke. Egal, wohin man das rote Rechteck bewegt, sieht es anders aus, d. h. dieser Bereich ist Unikat. In diesem Beispiel gilt die Ecke als gutes Merkmal in einem Bild. Daraus folgen die Hauptaufgabe und die Notwendigkeit des Algorithmus darin, derartige Merkmale zu finde.
Mit der Frage, welche Bestandteile für den Output im Allgemeinen generiert werden sollen, lässt sich die Antwort wie folgt formulieren: Die resultierenden Merkmale sind Teilmengen des Bildbereichs, entweder in Form von isolierten Punkten, kontinuierlichen Kurven oder verbundenen Regionen. Die Wahl hängt je nach Algorithmus ab. Im Fall der isolierten Punkte enthält das Merkmal die Bildkoordinaten des Bildmerkmals. Diese werden für die Weiterverarbeitung zur Positionierung und aktiven Positionsbestimmung des Trackings verwendet. Die erkannten Punkte, die später zur Positionierung verwendet werden, kann man auch als Ankerpunkte bezeichnen, da sich das Koordinatensystem daran „festhalten“ kann.
Abhängig davon, ob von einem markerlosen oder markerbasiertem AR System die Rede ist, wird entweder eine Position relativ zu einem Marker oder eine bestimmte absolute Raumposition in Abhängigkeit der Merkmale der Umwelt ermittelt. Im Falle der markerbasierten Abstimmung muss vorher eine Festlegung für das zu erkennende Muster erfolgen. Dies erfolgt mittels einer Referenzgrafik, dessen Merkmalspunkte man mit denen der realen Umgebung abgleicht. Die Position wird dann mit entsprechenden Übereinstimmungen zugeordnet.

Objekt-/Merkmalstracking:
Unmittelbar nach der Merkmalserkennung folgt das Tracking der Punkte. Der Input für diesen Schritt sind die Merkmalspunkte, des initialen Bildes befinden, und das darauffolgende Bild.
Die Verarbeitung sollte nach der Logik erfolgen, dass man eine Bewegungssequenz von mehreren Objekten betrachtet, die von einer statischen Kamera betrachtet werden. Unter der Annahme, dass es kein bestimmtes 3D-Modell der Bewegung gibt, beschränkt sich das Problem auf die 2D-Projektion der realen 3D-Bewegung. Die Punkte müssen demnach in der Bildebene verfolgt und ihre sich ständig ändernde Position aufgrund des bewegten Bildes neuberechnet werden über der Zeit bzw. der Bildänderung. Wie zur Veranschaulichung in Abb. 5 zu sehen.

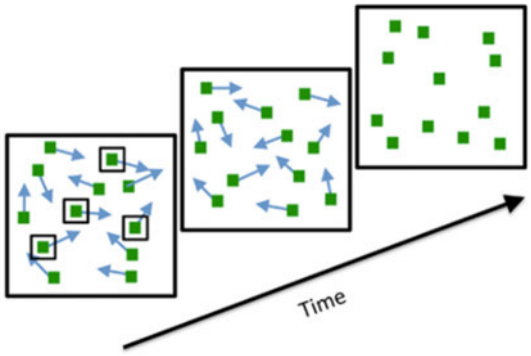


Abbildung 5 [5]

Den Output stellen die neuen Koordinaten dar, auf denen die Merkmalspunkte gemapped werden können.

Positionierung:
Um diesen Teil in seiner Funktion analysieren zu können, wird an dieser Stelle zuerst die Koordinatenfestlegung erläutert. Das aufgezeichnete Bild, dass es zu erweitern gilt, wird mit den vorangegangenen Methoden bearbeitet. Hieraus resultieren die genannten Trackingpunkte, dessen Position bereits als Bildkoordinaten gegeben ist. Diese Bildkoordinaten sind, aufgrund der Pixel Betrachtung der Merkmalserkennung, in Abhängigkeit der maximalen horizontalen und vertikalen Pixel Anzahl definiert. Definitionsgemäß ergeben sich daher kartesische Koordinaten.
Der Input der Positionierung sind die kartesischen Koordinaten der Trackingpunkte. Wie schon erwähnt, sind dieser veränderlich und stellen eine konstante Markierung der gefundenen Merkmale da. Die Abstraktion hierbei ist die zweidimensionale Änderung der Koordinaten, die aus der Projektion von aufgezeichneten Punkten eines 3D Raums hervorgehen.
Wie ist jedoch aus menschlicher Sicht lediglich mit einer 2D Darstellung eine Tiefe des Bildes erkennbar? Die dritte Koordinate entsteht, und damit auch die räumliche Wirkung, durch die konstante Überlagerung der Punkte mit den Pixeln, die den Raumpunkt darstellen und der Perspektive folgen. Folglich entspricht dieser Teil bereits, wenn man ihn Visualisieren würde, einer Erweiterung der Realität, da virtuelle Punkte einem zugrundeliegenden realen Punkt zugeordnet werden.

Die Punkte in der Realität werden fixiert damit anschließend das realitätserweiternde Objekt daran positioniert werden kann. Die initialen Bildpunkte können jedoch nicht unmittelbar nach der Auffindung erhalten bleiben, da sich das Bild letztlich bewegen soll und die Position zwangsläufig dynamisiert ist. Daher findet eine Verarbeitung statt bei der die fixierten Punkte verfolgt werden. Man spricht im Bereich der Computer Vision auch vom „Tracking“. Die endgültige Position für das Überlagern des virtuellen Objekts lässt sich in Abhängigkeit der Trackingpunkte definieren oder aufgrund einer Relation der Punkte zur Umgebung. Dabei muss auf geometrisch algebraische Weise eine Ermittlung der Objekt Position erfolgen. Mit jeder Bildbewegung wird dieser Schritt wiederholt, um die Position in einem konstanten Bezug zu den Trackingpunkten zu halten. Dies Erzeugt die Illusion, dass das Objekt im Raum fest verankert. Zusätzlich zur zweidimensionalen translatorischen Bewegung benötigt man eine Ermittlung der Rotation um die virtuellen Objektachsen. Letztlich sorgt diese Bewegung für die endgültig real wirkende Erweiterung. Der Betrachter sieht eine der realen Intuition folgenden Bewegung und bekommt das Gefühl sich mit der Kamera, um das Objekt als Blickzentrum zu bewegen. Wie schon erwähnt kann die translatorische Komponente durch das 2D- Tracking realisiert werden. Die Rotationen bzw. Blickwinkel Änderungen müssen aufgrund mehrerer Einflüsse berechnet werden. Als letzten Bestandteil des AR Systems ergibt sich infolgedessen eine relative Kamerawinkelbestimmung. Basierend auf diesem Winkel kann der Blickwinkel auf das Objekt und somit die Rotation um die drei Raumachsen bestimmt werden. Zum Output gehören somit abhängig von der Nullposition kartesische Koordinaten und Winkel um alle drei Raumachsen.

Darstellung:
Die Vollendung des AR Erlebnis ist die optische Überlagerung und damit Darstellung der Daten des virtuellen Objekts, nachdem alle Positionsinformationen gegeben sind. Die dafür einzusetzende Darstellungsfunktion sollte Positionsdaten und referenzierte Objektdaten erwarten (**Input**). Eine geeignete Lösung sorgt für das Anpassen der Bildpixel an den zu substituierenden Stellen (**Verarbeitung**). Den Vorgang kann man auch als überschreiben der aufgezeichneten Bildpixel der realen Umwelt mit neuen Informationen bezeichnen. Letztlich ist ein geändertes originales Bild auszugeben, um es dem Betrachter zu präsentieren (**Output**).

2.3 Lastenheft mit Zielsetzung für eine prototypische Lösung in Matlab

Die zugrundeliegende Arbeit dient als Grundlagenforschung und zur Untersuchung der Realisierbarkeit eines einfachen Prototyps mithilfe der Möglichkeiten der Software Matlab der Firma MathWorks Inc., dessen vielfältige Auswahl an Toolboxen einen Einstieg in das Thema Augmented Reality ermöglichen. Eines der Toolboxen besitzt viele der gängigen Computer Vision Algorithmen. In Anbetracht dieses Hintergrunds, soll diese Entwicklung ein Einstieg für weitreichendere Untersuchungen und Entwicklungen, die auf den rudimentären Bestandteilen eines AR Systems aufbauen können, vereinfachen.

Zielsetzung:
Ziel dieser Arbeit ist es eine Augmented Reality-Umgebung zu entwickeln, die in der Lage ist ein beliebiges, zuvor referenziertes CAD Objekt in einer AR-Umgebung darzustellen und somit optisch mit der realen Umgebung zu überlagern.
Um diesen Endzustand zu erreichen, soll in einer zweistufigen Entwicklung zunächst eine 2D Fotomontage und darauf aufbauend eine 3D Darstellung eines CAD Modells erzielt werden.
Das Entwickeln der Anwendung soll von einer technischen Dokumentation begleitet werden, die als Basis für ein Lernprogramm dienen kann und die Möglichkeiten mit der Entwicklungsumgebung von Matlab aufzeigt.
Innerhalb des Rahmens dieser Bachelor Thesis soll in mindestens weniger als 12 Wochen eine Lösung entstehen.

Einsatz:
Die Anwendung soll über die Matlab Software, die sowohl auf handelsüblichen PC oder Mobilgeräten verfügbar ist, ausführbar sein. Hierfür wird in beiden Fällen eine Kamera, inklusive der notwendigen Treiber, mit einer Auflösung von mindestens 720p und einer Bildrate von mindestens 30 fps vorausgesetzt. Eine Bildrate von mehr al 60 fps wäre jedoch bei Verfügbarkeit zu bevorzugen. Die AR Anwendung wird in einer ausreichend beleuchteten Umgebung ab einer Beleuchtungsstärke von 200 lx (gleicht einer üblichen Wohnraumbeleuchtung) optimal einsetzbar sein.

Funktionale und Anforderungen:

Nach dem Starten der Anwendung, soll die Umgebung in einer **bewegten Aufnahme** wiedergegeben werden.
Währen der Aufnahme soll eine **Erkennung** in der Umgebung und eine automatische Initialisierung stattfinden, sobald dies basierend auf dem Inhalt der aufgezeichneten Umgebung möglich ist.
Die **Initiale Position** wird gespeichert, um während der weiteren Aufzeichnung, später, verfolgt ("getracked") zu werden. Mit den Punkten wird die Positionierung und damit der Ausgangspunkt zu Laufzeitbeginn ermittelt.
Die besagte Position des Objekts soll stets bei einer Bewegung der Kamera wiedergefunden und erst mit Laufzeitende oder manuellem Abbruch gelöscht werden. Diese Funktion übernimmt das **Tracking**. Somit findet, in Abhängigkeit davon, eine automatische **Berechnung der neuen Positionierung** statt, die an die Darstellung übergeben werden kann.
Die Anwendung soll im Endzustand einen auswählbaren **CAD Datensatz** darstellen können. Die Daten erhält man dabei aus einem Zugriff auf einen **Server**, der die Informationen bereitstellt. Dies geschieht über einen **Client**, der die Daten anfordert. Auf einem Dateisystem, das somit erreicht werden kann, sollen die CAD Daten zur Verfügung abgelegt sein und zur Verfügung stehen.
Die Darstellung erfolgt in einer **Echtzeit Überlagerung** der aufgezeichneten Realität und zeigt ein virtuelles Objekt derart, als wäre es im realen Raum positioniert. Gleiches gilt für eine kurzzeitige Bildunterbrechung oder unzureichende Bilddaten. Dem Anwender soll eine kontinuierlich im Raum befindliche virtuelle Positionierung und Darstellung des 3D Objekts vermittelt werden.

Nichtfunktionale Anforderungen:
Die Bedienbarkeit soll auf einfachste Weise ohne tiefgehende informationstechnische Vorkenntnisse geschehen. Die Interpretierbarkeit und individuelle Anpassungen oder Optimierungen bedarf einem Grundverständnis für Informatik. Eine Aneignung von Grundlagen in der Programmiersprache von Matlab wird daher vorausgesetzt.
Die Anwendung soll **erweiterbar** sein und interessierten Lesern einen beschleunigten **Einstieg in die AR Entwicklung** geben, um optimierte, neue Anwendungen zu entwickeln, die aus den gewonnenen Grundlagen hervorgehen.
Da diese Arbeit auf die Entwicklung eines Prototyps abzielt, wird der Anspruch an eine experimentelle Forschungsanwendung gestellt, die dem Zweck der **Bildung und Hilfestellung für weitergehende Untersuchungen** dient. Dementsprechend handelt es sich auch um eine nicht kommerzielle Anwendung.

Phasen:
Die Entwicklung ist in drei Phasen zu unterteilen:
Die erste Phase, die **Grundlagen Untersuchung**, dient der Untersuchung und Vorstellung von konkreten, möglichen Algorithmen, die den notwendigen Bestandteilen und Anforderungen gerecht werden. Dokumentiert wird die Phase in Abschnitt 2.4 und 2.5.

Als zweite Phase lässt sich die **Synthese und Analyse von Lösungsmöglichkeiten** definieren (Abschnitt 2.6), in der, mithilfe der untersuchten Funktionen und deren Grundlagen, Lösungskonzepte entwickelt werden sollen.
Die dritte Phase ist die **Bewertung und Konzeptfestlegung** (Abschnitt 2.7), mit der die entwickelten Lösungen differenziert betrachtet werden und eine, zu entwickelnde, Lösung gefunden wird.
Mit der vierten Phase, der **Umsetzung**, wird das erarbeitete Konzept realisiert und die Schritte während der Entwicklung dokumentiert. Falls nötig werden, unter ständiger Berücksichtigung der Anforderungen, Anpassungen und intermediäre Lösungen zum Erreichen des Endzustands des Prototyps erarbeitet und dokumentiert.

2.4 Abgleich der Anforderungen mit verfügbaren Algorithmen und Lösungen

Mit den feststehenden Anforderungen und den erörterten allgemeinen Bestandteilen und Funktionsweisen einer AR Anwendung, gehört zur Untersuchung von Grundlagen im Detail, eine vorherige Untersuchung von vorhandenen Lösungen. Daher werden in diesem Abschnitt bereits verfügbare Algorithmen in Matlab, unter anderem aus dem Bereich der Computer Vision Toolbox, und einige AR Lösungen identifiziert und verglichen. Zunächst werden, im Folgenden, bereits existierende AR Lösungen und deren Ansätze präsentiert und verglichen. Dieser Abschnitt soll allumfassend, im Voraus, die grundsätzlichen Möglichkeiten von AR und die aktuelle Entwicklungsrichtung in der Forschung dieses Gebiets aufzeigen.

Wie Eingangs bei der Vorstellung der allgemeinen Funktionsweise beschrieben, gibt es eine große Unterscheidung bei AR Systemen. Im Fall des markerbasierten AR Systems findet eine Initialisierung durch ein Referenzobjekt statt. Im Wesentlichen handelt es sich bei dieser AR Methode um eine Abhängigkeit zu einem bestimmten, in der Realität befindlichen Objekt. Dieses befindet sich in der menschlich wahrnehmbaren Realität, beispielsweise als 2D Ausdruck in Papierform oder Fotodruck.

Möglich sind jedoch auch komplexere 3D Objekte als eine Art Auslöser [6]. Da das aufgezeichnete Bild eine 2D Darstellung der Realität ist, werden bei dieser Methode **Bild-Merkmal-Erkennungsalgorithmen** verwendet. Mit ihnen lassen sich Muster, auch „Interessenpunkte“ genannt, auf der aktuellen Aufnahme identifizieren. Dies setzt vor allem einmalige optische Eigenschaften des Objekts voraus und eine Hinterlegung der Eigenschaften des Referenzobjekts, um diese den erkannten Merkmalen zuzuordnen. Durch die Erkennung von Merkmalen und deren Position, wird in Abhängigkeit dieser, die Position des virtuellen Objekts bestimmt. Dabei fällt auf, dass hier eine Sicht Abhängigkeit vorliegt. Das Muster muss für eine Merkmal-Feststellung erkennbar bleiben, da sich diese Methode rein visuell orientiert. Es existieren jedoch für die Identifikation von Interessenpunkten, unterschiedlichste mathematische Ansätze aus den Bereichen der Algebra, Analysis, und Topologie. Einige der am weitest verbreiteten und gängigsten Algorithmen im Bereich des maschinellen Sehens für **die Erkennung von Bildmerkmalen** sind folgende [7]:

- **SIFT** (Maßstabsunveränderliche Merkmalstransformation),
- **SURF** (beschleunigt stabile Merkmale),
- **FAST** (Merkmale aus beschleunigter Bereich Testung),
- **ORB** (orientierter FAST und rotierender BRIEF Algorithmus)
- **BRISK** (binäre, stabile, unveränderliche, skalierbare Merkmalspunkte)

Die verglichenen Algorithmen sind beispielsweise in OpenCV verfügbar, einer open Source Bibliothek, die bereits entwickelte Algorithmen für „Computer Vision“ Lösungen zur Verfügung stellt.
Es existieren zahlreiche weitere Lösungen, jedoch sind die genannten Algorithmen in der Matlab Computer Vision Toolbox verfügbar und für diese Arbeit in Frage kommend. Die Funktionen lassen sich der Anforderung der **Erkennung** zuordnen [8], weil sie die Eigenschaft besitzen markante Bildmerkmale zu erkennen und deren Position auf dem Bild auszugeben. Auf die einzelnen Vor- und Nachteile der Algorithmen wird im Abschnitt 2.5.1 eingegangen.

Darauf aufbauend wird die **initiale Positionierung** möglich. Die Methode der Positionsbestimmung und Ausrichtung hängt von der Wahl der Erkennungsalgorithmen ab. Aus diesem Grund gibt es hierfür keine allgemein definierten Algorithmen für AR Anwendungen. Die Positionsbestimmung erfolgt in der Regel über einen analytisch geometrischen Ansatz, der auf den Output Daten der Erkennung basiert. Die genannten Algorithmen liefern erkannte Punkte, die in einem Koordinatensystem interpretiert werden können. Für die Transformation der Koordinaten zur gewünschten Initial Position sind zwei Ansätze bereits in einigen AR Systemen existent:

Eine auffindbare Lösung ist die Ausrichtung der Objekt-Koordinatenachsen in Abhängigkeit zur aktuellen Ansicht [9]. Dies entspricht einem rein visuellen Ansatz und kann in Matlab zum Beispiel durch eine projektive Transformation, mithilfe der Funktion **estimateGeometricTransform2D** bestimmt werden [10]. Die Funktion basiert auf dem RANSAC-Algorithmus [11]. Sie schließt Ausreißer mit Hilfe des „M-estimator sample consensus“ (MSAC)-Algorithmus aus. Der MSAC-Algorithmus ist eine Variante des Zufallsstichproben-Konsens-Algorithmus (RANSAC). Aufgrund der zufälligen Natur des MSAC-Algorithmus sind die Ergebnisse zwischen den Läufen möglicherweise nicht identisch.
Es gibt jedoch auch Lösungen mit weiteren Peripherie Daten, außer den optischen. So gibt es beispielsweise AR Systeme, die eine Kombination aus optischen, piezoelektrischen oder auch induktiven Sensoren verwenden [12]. Hier wird die relative Kamera Ausrichtung zum virtuellen Objekt durch die Winkel der Raumachsen des Sensors eines mobilen Endgeräts, z.B. Smartphone, bestimmt. Mit dieser Lösung wird auf visueller Basis lediglich die Position des Ursprungs auf der Bildebene bestimmt und z.B. mithilfe des Neigungssensors die Ausrichtung.

Für das **Tracking** der Ursprungsposition existieren bereits ebenfalls stark verbreitete Algorithmen, die der Anforderung eines **Trackings** gerecht werden können. Diese sind auch über die

Plattform OpenCV oder Matlab zugänglich. Um sich auf die Entwicklungsumgebung von Matlab zu beschränken, sind dort folgende Algorithmen als Funktionen verfügbar [13]:

- **HistogramBasedTracker/CAMShift -Algorithmus** [14]
  (Verwendet das Histogramm der Pixelwerte, um das verfolgte Objekt zu identifizieren)
- **PointTracker/Kanade-Lucas-Tomasi (KLT) – Algorithmus** [15]
  (Berechnet die Translation von erkannten Merkmalspunkt-Informationsobjekt („feature points"), die zuverlässig verfolgt werden können)
- **BlockMatcher** [16]
  (Bestimmt die Bewegung zwischen zwei Bildern)
- **TemplateMatcher** [17] (Bestimmt die Position der treffendsten Übereinstimmung zu einer Vorlage)

Auf die Funktionsweise und Eigenschaften eines geeigneten Trackingalgorithmus soll in 2.5.2 eingegangen werden.

Für die **Berechnung der neuen Positionierungen** können ebenfalls die Lösungen für initiale Positionierungen verwendet werden. Für diese Anforderung lässt sich die ebenselbe Funktion, **estimateGeometricTransform2D** verwenden, falls ein rein visueller Ansatz verfolgt wird. Die Berechnung basiert in beiden Fällen auf der Relation zur Ausgangslage. Die beiden Anforderungen der Positionierung unterscheiden sich lediglich dadurch, dass sie zum einen (Erkennung) vor der Verfolgung und zum anderen während der Verfolgung (Tracking) die Positionen des virtuellen Objekts bestimmen. Zwangsläufig ist jedes AR System derart aufgebaut, dass es während der Verfolgung eine Transformation vollzieht, da sich die Position ständig ändert, und nur im Ideal statisch wäre. Somit ist die Reihenfolge des Trackings mit darauffolgender neu Positionierung in jedem AR System unabdingbar. Bei einem Teil oder nicht visuellen Ansatz für die neu Positionierung existiert über die Software Matlab Mobile [18] die Möglichkeit mit einem Add-On auf weitere, außer den fotosensorischen, Hardwaredaten in Echtzeit zuzugreifen.

Die **Darstellung** erfolgt in vielen Anwendungen mittels eines Grafik Renderings, dass polygonale Flächen erzeugt oder aber auch ein Gitternetz, die als einen 3D Körper bilden. Bei zweidimensionalen virtuellen Objekten, wie einem Bild oder Video wird die Grafik auf das originale Bild überlagert und die Bildpunkte substituiert. Das Bild wird je nach Perspektive skaliert und verzerrt. Die Lösungsmöglichkeiten bezogen auf diese Arbeit jedoch, beschränken sich auf die Generierung von Flächen bzw. sogenannten Patches, die in Matlab mit der Funktion **patch** erzeugt werden können [19]. Alternativ ist auch ein Polygonnetz möglich, welches anschließend gefüllt wird um eine kontinuierliche Oberfläche zu erreichen [20]. Um die Informationen für das 3D Rendering zu erhalten können Algorithmen verwendet werden, die entsprechende CAD Datensätze in Patches konvertieren, die eine Vereinfachung der Außen Geometrie darstellen. Eine Software namens „3D_Evolution_Simplifier" der Firma „coretechnologie" bietet hier eine Lö-

sung an [21]. Für Matlab gibt es keine erwerbliche Funktion für diese Anforderung. Jedoch existieren open source Lösungen innerhalb der Matlab Foren, die von Nutzern geteilt wurden und beispielsweise einen STL Datensatz lesen und als Daten für die Patch Funktion in Matlab übergeben werden können [22].

Bei Beachtung der **markerlosen** AR Systeme fallen folgende Unterschiede zur Lösung der Anforderungen auf. Die Initialisierung funktioniert ohne weitere reale Objekte. Eines der eingängigsten Beispiele ist die Ebenen-Erkennung. Bei dieser Methode findet eine Erkennung einer vorliegenden ebenen Fläche statt, die als Orientierung für die Objektausrichtung dient [23]. Voraussetzung hierfür ist eine planare Oberfläche, die mit einer Ebene gemittelt werden kann (s. Abb. 6).

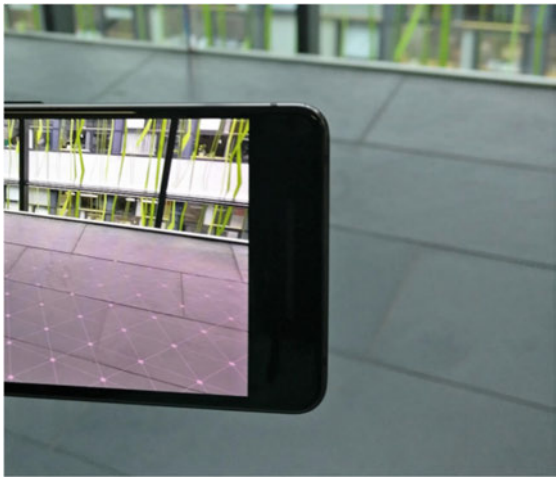


Abbildung 6 [42]

Sobald eine Ebene ermittelt wurde wird das Koordinatensystem auf einem Punkt in der Ebene verankert. Ein fortgeschrittenes Beispiel für diese Umsetzung ist das AR System von Apple Inc., namens „Apple ARKit“ [24]. Um bei diesem Beispiel zu bleiben, funktioniert die Ebenen-Erkennung in der Art, dass der Raum, der mit der Kamera aufgezeichnet wird, mittels SLAM (**S**imultaneous **L**ocalization **a**nd **M**apping; *z.Dt. „Simultane Positionsbestimmung und Kartierung“)* erkannt wird.

Abbildung 7 [25]

Bei dieser Methode wird das Objekt einmalig positioniert und durch die Abtastung des Raumes als 3D Model in Relation gesetzt. Die Bewegung im Raum kann somit dem **Tracking** von Inte-

ressenpunkten des aufgezeichneten Bildes an der Stelle des Raumes zugeordnet werden. Die virtuelle Sicht, wird damit **neu berechnet** und gezielt zur **neuen Position** navigiert. Daher würde das Objekt immer wieder an der gleichen Stelle gefunden werden, wenn die Kamera die Sicht vom Ursprungspunkt abwendet und anschließend wieder zur Ausgangslage zurückkehrt. Dieser Ansatz bietet aktuell die höchste Genauigkeit bei markerlosen AR Anwendungen.
SLAM existiert in verschiedensten Variationen. Das Prinzip setzt lediglich eine eindeutige räumliche Ermittlung voraus. Je nach Präzisionsniveau der AR Anwendung bzw. in welchem Bereich sie eingesetzt werden soll, werden die Lösungen eingesetzt. So wird unter anderem bei Outdoor-AR Apps, die einen größeren Maßstab ermöglichen, eine Ortsbestimmung per GPS verwendet. Bei einer Betrachtung in einem Abbildungsmaßstab < 1 wird optisches und kinematisches Tracking, basierend auf lokalen Sensordaten, verwendet.

2.5 Grundlegende Funktionsweisen von essentiellen Algorithmen

2.5.1 Objekterkennungsalgorithmen

Ein wichtiger initialer Bestandteil bei der Realisierung einer Augmented Reality Umgebung stellt die Objekterkennung dar. Als Objekt kann man hier, bezogen auf eine vorhandene Umgebung, die markanten Merkmale einer Bildaufnahme verstehen. Diese Merkmale werden erkannt und als Objekt gespeichert. Sie sind von Bedeutung um eine Verankerung als initiale Position daraus abzuleiten. Diese Verankerung lässt sich auch als Identitätszuordnung verstehen (s. Abb. 8). Die Erkennung ist zu Beginn der Laufzeit eines jeden beliebigen AR Systems notwendig und unabhängig von der AR Methode. Jedoch erfolgt hier eine ausschließliche Fokussierung auf optisch punktbasierte Objekterkennungen.

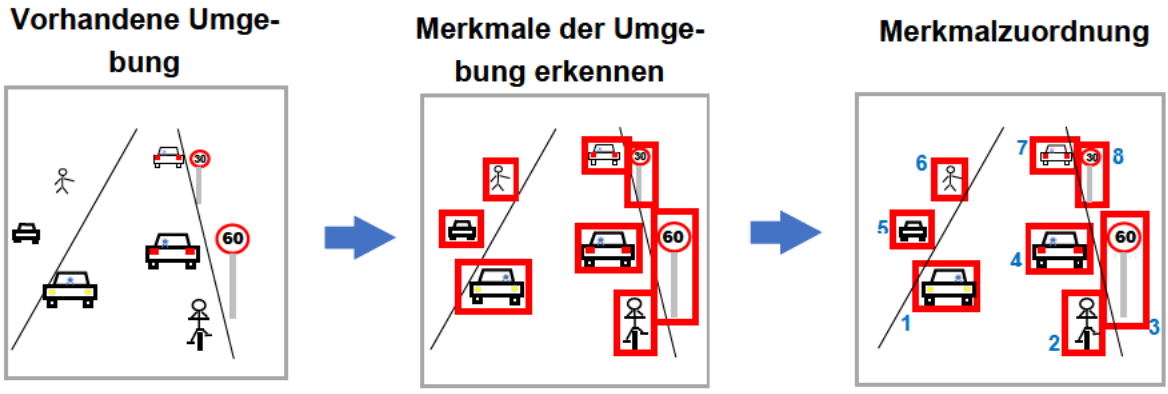


Abbildung 8

2.5.1.1 Testreihe zur Bewertung und Auswahl eines geeigneten Objekterkennungs-Algorithmus

Im Folgenden werden die Ergebnisse einer Untersuchung zu ausgewählten optisch punktbasierten Objekterkennungsalgorithmen, auch „point based Feature Detectors“, die in der MATLAB Computer Vision Toolbox zur Verfügung stehen vorgestellt und anhand der Ergebnisse, einer eigenständig durchgeführten Testreihe für den hier zu untersuchenden Anwendungsfall einer AR

Umgebung bewertet. Zur Bewertung und Auswahl einer geeigneten Implementierung in Matlab werden vier der Funktionen, aus 2.4, getestet. (Der SIFT Algorithmus ist nahezu identisch bzw. ähnlich zum SURF Algorithmus und wird daher nicht mit getestet. Zumal es keine direkte Funktion für diese in Matlab gibt.) Das Matlab Skript, in dem die Funktionen getestet werden, befindet sich in der Anhang A unter der Bezeichnung **test_detection.m**. Ebenso sind darunter alle Messergebnisse enthalten, sowie die sechs Aufnahmen als Testobjekte.

Durchführung:
Für alle fünf Testreihen gelten gleiche Durchführungsbedingungen:
Bei der verwendeten Kamera handelt es sich in allen Fällen um die gleiche Hardware. Es wird eine Bilderreihe von sechs Aufnahmen mit der Abbildung eines durchgehend, identischen Musters durchgeführt. Diese Bilder werden als Input Daten verwendet und passend zur Funktionsparametrisierung eingesetzt, um einen vergleichbaren Output über alle Versuche zu erzeugen. Es wird in folgenden Eigenschaften D gemessen**:**

Laufzeit:
Die Zeit D_t in Sekunden (s)**,** die bis zur Vollendung des Outputs vergeht. (arithmetisches Mittel über alle sechs Aufnahmen).

Anzahl:
Mittlere Anzahl D_n der erkannten Merkmale/Objekte die ausgegeben werden. (arithmetisches Mittel über alle sechs Aufnahmen).

Rotationsinvarianz:
Stabilität der Wiedererkennung von Merkmalen gegenüber Rotationen des Musters. Dabei wird die Stabilität mit der Standardabweichung der erkannten Punkte gemessen D_{σ_n}. (Für die Verdrehung und anschließende Berechnung werden die letzten drei Bilder, die immer um 45° weiter verdreht werden, zugrunde gelegt)

In den Grafiken auf den nachfolgenden Seiten sind die Ergebnisse des Auswertungsskripts dargestellt:

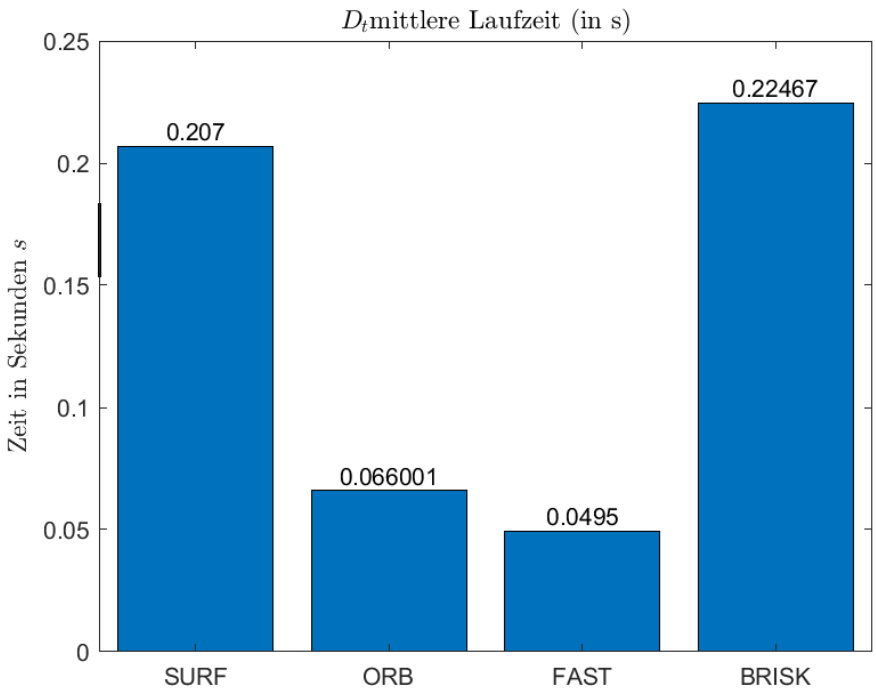


Abbildung 9

Die mittlere Laufzeit ist bei der FAST Funktion am besten und liegt nahezu gleichauf mit ORB. Dies wäre ein Vorteil bei bewegten Aufnahmen und lässt die FAST Methode eindeutig als beste Variante in der Kategorie Laufzeit hervorgehen.

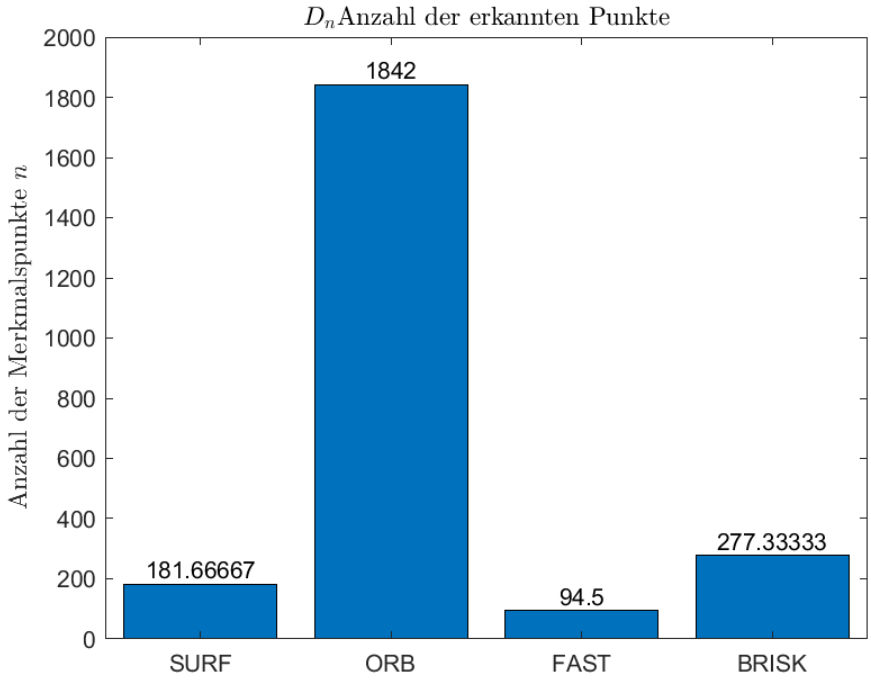
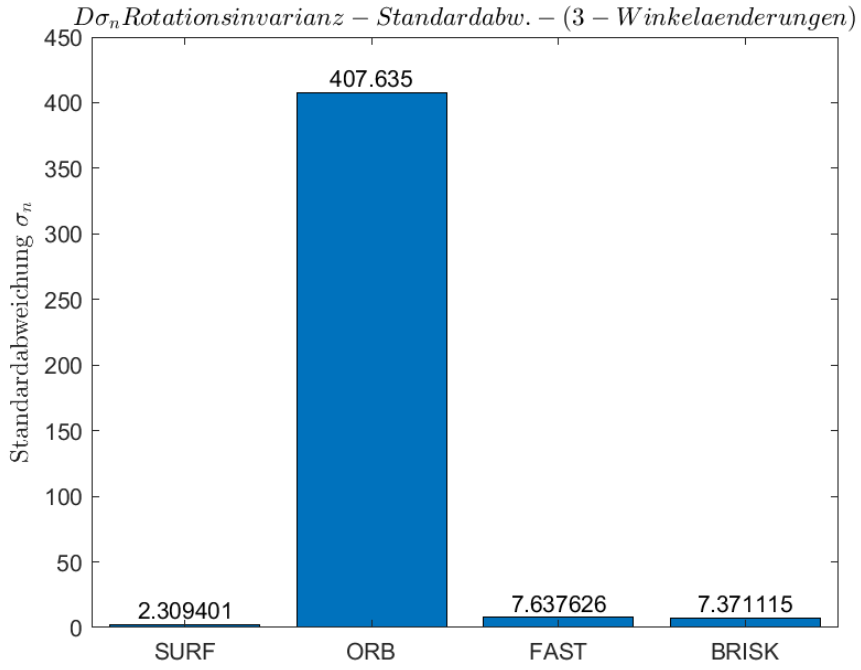


Abbildung 10

Die mittlere Anzahl der erkannten Punkte ist bei ORB mit Abstand am höchsten, jedoch scheint es so als würden in zu kurzer Zeit zu viele Punkte erkannt werden, sodass eine zuverlässige Merkmalserkennung fraglich ist.

Abbildung 11



Beim Vergleich der Rotationsinvarianz fällt deutlich auf, dass ORB negativ hervorsticht. Damit lässt sich in Verbindung bringen, wie geeignet die große Anzahl, der erkannten Punkte, tatsächlich zu sein scheint. Daraus würde sich keine Empfehlung für ORB für bewegte Bilder ergeben.

In der untenstehenden Matrix können alle Kategorien quantitativ zu einer Gesamtwertung mit der danebenstehenden Formel zusammengefasst werden. Dabei wird die Standardabweichung auf die Anzahl n normiert und gewertet, sodass für steigende Abweichungen eine niedrigere Bewertung entsteht. Ebenso wird die Laufzeit invertiert gewertet, da eine kürzere Laufzeit tendenziell besser ist und höher bewertet wird. Die Summe der Grundgesamtheit über der Standarbweichung, bei einer vergleichbaren Rotation, und der reziproken Laufzeit ergibt die Gesamtwertung für jeden einzelnen Algorithmus innerhalb der Matlab Funktion.

D	SURF	ORB	FAST	BRISK
n	181,6667	1842	94,5	277,33333
t	0,207	0,06601	0,0495	0,22467
σ	2,309401	407,635	7,637626	7,371115
∑	**83,4949091**	**19,667968**	**32,574975**	**42,075312**

$$D = n * \frac{1}{\sigma} + \frac{1}{t}$$

Aus diesem quantitativen Vergleich ergibt sich für die Auswahl des, hier am besten zu bewertenden, Algorithmus der **S**peeded-**U**p **R**obust **F**eature Algoritmus, der unter dem Akronym SURF weit verbreitet und patentiert ist. Trotz der Ähnlichkeit zum SIFT Algorithmus liegt der Vorteil gegenüber diesem in der zeitunabhängigen Skalierung des Filters für die Mustererkennung. Statt einem Mittelwertfilter verwendet SURF einen **Gauß-Filter**, dessen Einsatz neben der

allgemeinen Funktionsweise des Algorithmus im nachfolgenden Kapitel, 2.5.1.2, erläutert wird. In Matlab ist dieser Algorithmus durch die Funktion „**detectSURFFeatures**“ umgesetzt und in der Computer Vision Toolbox erhältlich [26].

2.5.1.2 Funktionsweise und Vorstellung des gewählten Objektkennungs-Algorithmus

Der SURF Algorithmus stellt sich im praktischen, anwendungsspezifischen, Vergleich aller Objekterkennungsalgorithmus als die beste Lösung heraus. Im Nachfolgenden wird, zum besseren Verständnis, für eine optimale Integration der Funktion in das zu entwickelnde Skript, auf die Bedeutung und Funktionsweise des Algorithmus eingegangen. Die Erkennung spielt in einer AR-Anwendung eine zentrale Rolle, da von ihr die Qualität der nachfolgenden Prozessschritte im gesamten Anwendungsalgorithmus abhängt:
SURF wurde erstmals von Herbert Bay, Tinne Tuytelaars und Luc Van Gool veröffentlicht und auf der European Conference on Computer Vision 2006 vorgestellt. Eine Anwendung des Algorithmus ist in den USA patentiert [27]. Der SURF-Algorithmus arbeitet, allgemein beschrieben, mit einer Approximation, basierend auf einer Hesse-Matrix. Hinsichtlich der Hesse-Matrix gelten folgende Grundlagen für das Prinzip der Anwendung, in der Bildverarbeitung.

Im Allgemeinen lässt sich damit die Approximation von mehrdimensionalen Funktionen vollziehen, indem alle möglichen Ableitungskombinationen als jeweilige Elemente enthalten sind. Das bedeutet, im Fall der Bildverarbeitung, eine Eignung für die Verwendung von Integralbildern, nach Paul Viola und Michael Jones [28]. Hierbei summiert man die Pixelintensitäten innerhalb von Bereichen eines Bildes, die durch ein Rechteck begrenzt werden. Integralbilder bieten unabhängig von der Größe des Bildes oder eines momentanen Bereiches den gleichen Berechnungsaufwand und gehen daher bei der Skalierung der Bildgröße mit einer konstanten Laufzeit einher. Die, daraus resultierende, schnelle Berechnung, erklärt, unter anderem, bereits einen Vorteil dieses Algorithmus.
Im speziellen wird die Hesse-Matrix hier mit einfachen Box-Filtern der Größe 9x9, zeitunabhängig grob approximiert. Um das Bild quantitativ verarbeiten zu können werden den Pixeln Intensitäten zugeordnet. Die Intensitäten der Pixel werden mit der Helligkeit beschrieben. Folglich wird keine Farbinformationen benötigt und das Bild als Graustufenbild, mit nur einem Kanal, verarbeitet. Die niedrige Rechenleistung liegt darin begründet, dass die Summe der Pixel des Gesamtbildes lediglich mittels der Werte aus jeweils 4 Pixeln des Integralbildes berechnet werden kann. Im Beispiel der Abb. 12, nach der Methode „Summed-Area Tables for Texture Mapping“ [29], bräuchte man für die Summe, beschrieben durch Fläche „IV“, lediglich vier Speicherungen $(1,2,3,4)$und drei Additionen $(4 + 1 - 3 - 2)$.

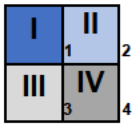


Abbildung 12

$$I_{\Sigma}(\boldsymbol{x}) = \sum_{i=0}^{i \le x} \quad \sum_{j=0}^{j \le y} I(i,j) \qquad F1$$

Mit der Faltung über die Integralbilder können somit wichtige Bildpunkte approximiert werden. Unter der Faltung, auch Konvolution, versteht man einen Operator der eine resultierende Funktion für zwei Funktionen (meist Eingangs- und Ausgangsfunktion) liefert. Die Summe der Produkte, bzw. das Integral über einen diskreten Definitionsbereich bildet das Ergebnis/die Faltung der beiden Funktionen.

$$(f * g)(n) = \sum_{k \in D} f(k)g(n-k). \qquad F2$$

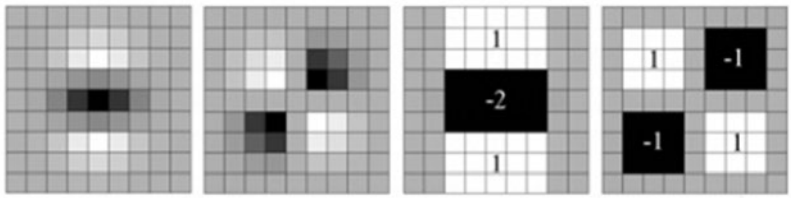
Besonders in der digitalen Signalverarbeitung wird aufgrund von diskreten Werten, wie zum Beispiel bei Pixeln, „gefaltet“, um einen sogenannten Faltungsfilter zu bilden. Durch einen Filterkern auch Faltungsmatrix genannt, werden die Kontrastwerte miteinander verrechnet. Die Aufgabe der Filterung, mittels der Faltung, ist die Abgrenzung von Ausprägungen des Frequenzspektrums einer Funktion (in diesem Falle einer Bildfunktion), durch eine Hervorhebung oder Verminderung dieser. Dieser Filterprozess dient im Beispiel des SURF Verfahrens zur Reduzierung des Bildes auf die markantesten Bildpunkte. In diesem Verfahren ergeben sich für die Faltung der zweiten Ableitung der Gauß-Funktion $\frac{\partial}{\partial x^2}$ g(σ) mit der Bildfunktion $I(x)$ im Punkt x , die Elemente $L_{yy}(x,\sigma),\ L_{xy}(x,\sigma),\ L_{xx}(x,\sigma),\ L_{yy}(x,\sigma)$ der Hesse-Matrix und damit wiederum die Filter. Sie stellen die Kombinationen der ersten und zweiten Ableitungen der Bildfunktion x^2, $y^2,\ xy,\ yx$ dar.
Im Frequenzraum sorgt der Gauß Filter für eine Hervorhebung von größeren Frequenzänderungen. Übertragen auf die Bildverarbeitung bedeutet dies eine Hervorhebung der kontrastreicheren Bereiche an Pixeln. Demnach werden **Werte über die Kontrastverteilung,** bestimmt, die eine Indizierung der Lichtintensität widerspiegeln. Umsetzbar sind hierbei Normierungen bezogen auf die Gesamthelligkeit eines Bildes. Wobei nur der Betrag zum Falten des Bildes, wobei gilt $abs(yx) = abs(xy)$, benötigt wird. Zusätzlich kommt die Standardabweichung zur Skalierung der Gauß Funktion zum Tragen. Während ein höheres Sigma einen größeren, zu berücksichtigenden Bereich definiert. Mithilfe von Sigma kann sozusagen der Abbildungsmaßstab bezogen auf das auszuwertende Bild eingestellt werden [27].

$$\mathcal{H}(\mathrm{x},\sigma) = \begin{bmatrix} \mathrm{Lxx}(\mathrm{x},\sigma) & \mathrm{Lxy}(\mathrm{x},\sigma) \\ \mathrm{Lyx}(\mathrm{x},\sigma) & \mathrm{Lyy}(\mathrm{x},\sigma) \end{bmatrix} \qquad F3$$

Durch die Bestimmung der Determinante, der Hesse-Matrix, kann man einem ganzen Bildbereich, der aus mehreren Pixeln zugleich besteht, genau einen Wert beimessen. Ein höherer Wert bzw. **größere Determinante det(H)** wird als **wahrscheinlicherer Keypoint** interpretiert und kommt als solcher in Frage. Als qualitative Analogie kann man diesen Wert mit dem optischen Gewicht gleichsetzen, das häufig als entscheidend für markante Punkte zählt, wenn man ein Bild in dessen einfachsten Kontrasten betrachten würde und lediglich die auffälligsten Punkte erkennbar bleiben, wie zum Beispiel 1. v.L. Abb. 13.

Unter der Berücksichtigung dieser Bestimmung der Determinante wird der Wert eines Bildbereichs an **diagonalen, horizontalen oder vertikalen Kanten gleich Null (det(H)=0)**, demnach sind dort auch keine Keypoints bestimmbar.

Zusätzlich, als Besonderheit des SURF Algorithmus, findet auch eine Approximation der zuvor bestimmten Filter statt. Diese ergänzende Vereinfachung, diskretisiert die Bereiche noch stärker. Die Approximationsfunktion von $\mathcal{H}$ vereinfacht, mittels der Variable s, die der Seitenlänge eines Filters zugeordnet ist, die Gauß-Filter zu 3x3 Filtern mit einheitlicher Verteilung auf die Bildpunkte. Darunter leidet jedoch die Präzision und es ebenso die Rotationsunabhängigkeit/Rotationsinvarianz. Dennoch ist diese weiterhin ausreichend vorhanden, was im darauffolgenden Abschnitt, zur Parametrisierung, noch weiter ausgeführt wird.



$$\mathcal{H}_{approx.}(x, s) = \begin{bmatrix} Dxx(x, s) & Dxy(x, s) \\ Dyx(x, s) & Dyy(x, s) \end{bmatrix} \quad F4$$

Abbildung 13 [27]

Zusätzlich wird zur Approximation der Faktor ω mit einem Wert von $\omega \sim= 0.912$ verwendet, um eine Gewichtung zu erzeugen, die für eine Angleichung an die ursprünglichen, nicht approximierten Filter sorgt, gem. [27] ergibt sich daher folgendes:

$$\det(H_{approx.}) = D_{xx}\, D_{yy} - (\omega D_{xy})^2 \qquad \omega \approx 0.912 \quad F5$$

Wie bereits erläutert wird die Determinante an Horizontalen, Vertikalen und Diagonalen Geraden gleich null. Demnach müssen zwei Winkel, die eine Ecke bilden durch das Muster entstehen, wie in den **Grafiken d bis h** zu sehen. In den Mustern **a** bis **b** wird die Determinante dementsprechend gleich null. Daher werden die meisten Keypoints bei Mustern/Markern mit vielen variierenden Winkeln und somit auch Ecken ermittelt. Aufgrund dieser Besonderheit entsteht eine Korrelation zwischen größeren Ecken (Am Beispiel in der Grafik g und h zu sehen) und größeren Filter, um bei gewissen Mustern stets eine Ecke in Relation zur Gesamtgrafik bestimmen zu können.

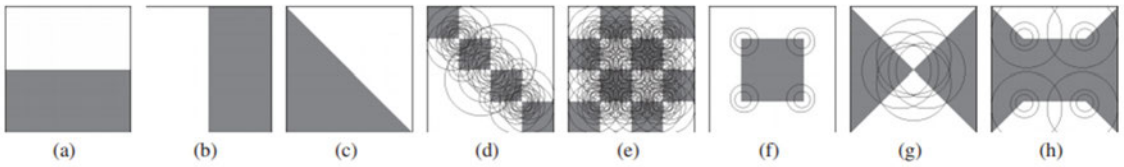


Abbildung 14 [30]

Die Größe der Approximationsfilter wird aus diesem Grund variiert, um sicherzustellen, dass essenzielle Bildpunkte unabhängig des Maßstabes der Grafik erkannt werden. Die Skalierung der Filtergrößen lässt sich in Oktaven einteilen. Jede Oktave enthält jeweils 4 Filtergrößen. Der erste und vierte Filter dient hierbei lediglich als Vergleichsmöglichkeit, bzw. als Grenzwertfestlegung, für die darauffolgende Abwägung der Relevanz von Bildpunkten. Damit eine ungerade Matrix mit diskretem Mittelpunkt für die Faltung beibehalten wird, beginnt die erste Vergrößerung mit 6 Pixeln pro Filter. Nach der ersten Oktave wird in der zweiten eine doppelte Vergrößerung verwendet. Dies setzt sich bis zur dritten Oktave, und somit einer ungefähren Skalierung mit dem Faktor acht, fort. Eine Vergrößerung darüber hinaus würde nicht zur Erkennung weiterer Bildpunkte führen. Die Skalierung ist in Abb. 15 nochmals geometrisch über die Anzahl der Teilflächen verdeutlicht.

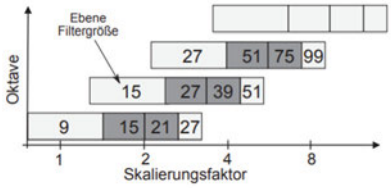


Abbildung 15 [30]

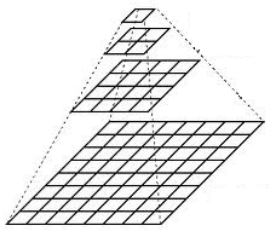


Abbildung 16 [30]

Mithilfe des erläuterten allgemein funktionalen Hintergrunds widmet man sich im Folgenden der Identifikation von besonderen Bildpunkten „interest points“ und der speziellen Parametrisierung der SURF Funktion in Matlab. Wie bereits erwähnt dient der Algorithmus der Quantifizierung von Bildmerkmalen, indem für Bildbereiche Kontrastwerte ermittelt werden. Um mit diesen Werten eine Selektion für markante Bildpunkte zu treffen, muss ein vordefinierter Grenzwert überschritten werden. Hierbei wird die Determinante der Hesse-Matrix mit dem definierten Maximalwert für diese (Parametername: *MaxThreshold*) verglichen. Gleichzeitig muss der Wert ein lokales Maximum in der 3 x 3 x 3 Umgebung sein, die jeweils durch den 1. Und 4. Filter in jeder Oktave gebildet werden (9x9 und 27x27; …; ...). Diese Überprüfung findet in jeder Oktave statt und wird somit dreimal durchlaufen. Da die Oktaven überlappen kann es hier zu doppelten Identifikationen kommen.

Sobald die Interessenpunkte gefunden wurden, werden die Punkte durch Interpolation der Bildkoordinaten (x, y) und der Skalierung σ genauer bestimmt, sodass dessen Bildkoordinaten, Gewichtung und der Maßstab ausgegeben werden können. *(Die Outputs lauten folgendermaßen: SURFPoints; location, Metric, Scale).*

Eine der wichtigsten Besonderheiten dieses Algorithmus ist die Rotationsinvarianz. Für das Erzielen dieser Eigenschaft wird jedem erkannten Punkte eine konstant abrufbare Orientierung zugeordnet. Auf einen 6sigma Radius, ausgehend vom Merkmalspunkt, werden Haar-Wavelet-Resonanzen in x- und y-Richtung, berechnet. Dabei wird die Orientierung mittels eines $\frac{\pi}{3}$ Kreisausschnitts ermittelt, der über den Kreis mit eingeschlossenen Resonanzpunkten iteriert, um für jeden Schritt einen resultierenden Vektor aus den Gewichtungen der Resonanzpunkte zu ermitteln. Der größte bestimmbare Vektor beschreibt die Orientierung des Merkmalspunktes. Diese Bestimmung der Ausrichtung eines Punktes ermöglicht die Robustheit gegenüber Rotationen eines zugrundeliegenden Bildes. Man nennt diesen Teil des Algorithmus auch „Descriptor“. Dieser wurde nicht bei jedem Erkennungsalgorithmus umgesetzt der verfügbar ist. Daher bietet sich die

Verwendung des SURF Algorithmus umso mehr an, da mit dessen Hilfe eine analytische Investition für diesen Abschnitt einer AR Umgebung am günstigsten ist und alle nötigen Prozessbestandteile einer Erkennung und Identifikation zuverlässig realisiert.
Somit enthält die Funktion in Matlab zusätzlich als Output, welcher Matlab intern auch als „*SURFPoints Object*" bezeichnet wird, auch die Orientierung, „*Orientation*". Insgesamt belaufen sich die wichtigsten Outputs, die in diesem Zusammenhang benötigt werden, als Speicherung in diesem Objekt, auf: Die Koordinaten, den Maßstab, die Gewichtung und die Orientierung (*Location, Scale, Metric, Orientation*). Das SURFPoints Objekt besteht daher aus mehreren Arrays und zum einen aus einer Mx2 Matrix, welches die x und y Koordinaten der gefundenen SURF Features enthält. Für eine genaue Zuordnung der Merkmalspunkte für nachfolgende Berechnungen oder einen späteren Abgleich werden sogenannte Merkmalsvektoren, auch Deskriptoren genannt, benötigt. Diese lassen sich mit der Funktion „extractFeatures()" aus dem mit „detectSURFFeatures" verarbeiteten Intensitätsbild berechnen und damit die zugehörigen Positionen extrahieren.
Die Funktion leitet die Deskriptoren aus den Pixeln ab, die einen interessierenden Punkt umgeben. Die Pixel repräsentieren und entsprechen Features, die durch die Position eines Punktes spezifiziert werden. Jeder Punkt gibt die zentrale Position einer Nachbarschaft an.

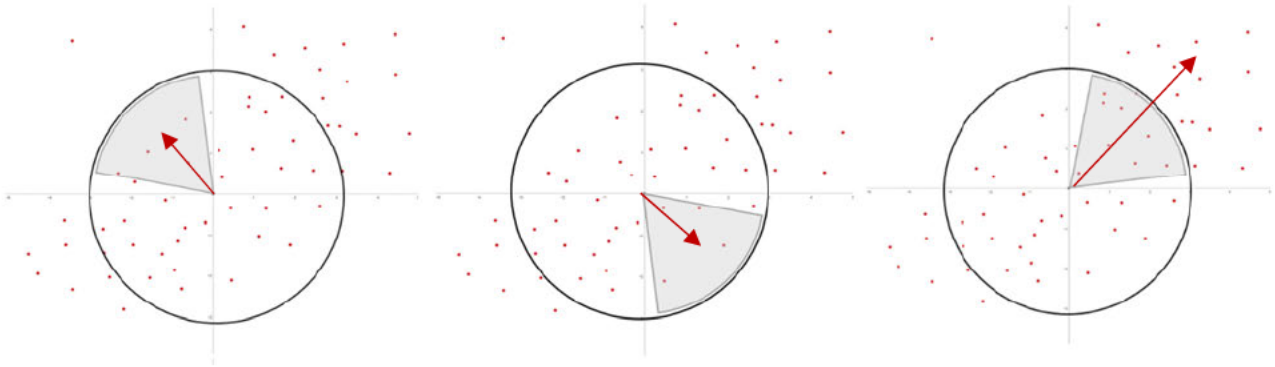


Abbildung 17

2.5.2 Objekttrackingalgorithmus

Zu Beginn sollte man den wesentlichen Unterschied und die Aufgabe eines Trackingalgorithmus festhalten. Unter dem Objekttracking versteht man die Verfolgung von bereits erkannten Objekten, die beispielsweise durch den im vorangegangen Objekterkennungsalgorithmus erkannt und identifiziert werden. Dabei ist das Objekttracking nicht an zweidimensionale Daten gekoppelt, sondern kann auch im Zusammenhang mit 3D Daten verwendet werden. Im Wesentlichen beseht die Funktion eines Trackingalgorithmus darin, zu jeder Bildänderung die neuen Bildkoordinaten eines zuvor bestimmten Objekts zu bestimmen. Dem Einzel Objekttracking und dem Multi Objekttracking. Angesichts der zugrundeliegenden Thematik einer AR Anwendung in dieser Arbeit beschränkt man sich im Folgenden auf das Tracking bei bewegten zweidimensionalen Aufnahmen.

Die meisten Tracking-Algorithmen arbeiten, nach dem Stand der Technik, mit einer Verfolgung basierend auf einer Erkennung bei dem sie zunächst Objekte in der Szene finden bzw. erkennen und dann die entsprechenden Positionen der Objekte in der nächsten Aufnahme ermitteln. Heutzutage sind die Detektoren außerordentlich leistungsfähig und auf unterschiedlichste zugrundeliegende Aufnahmen, parametrisch angepasst werden. Infolgedessen haben sie sich zum Standard-Input für Tracking-Algorithmen entwickelt.
Alle Trackingalgorithmen kann man übergeordnet in zwei Gruppen abstrahieren. Multi Objekt Tracking zielt darauf ab Videos zu analysieren, um Objekte zu verfolgen, die zu einer oder mehreren Kategorien gehören, ohne vorheriges Wissen über das Aussehen und die Anzahl der Ziele. In diesem Fall werden die Bewegungen aller erkannten Objekte unabhängig voneinander verfolgt. Die Hauptschwierigkeit bei der gleichzeitigen Verfolgung mehrerer Ziele ergibt sich aus den verschiedenen Verdeckungen und Wechselwirkungen zwischen Objekten, die manchmal auch eine ähnliche Ausprägung haben können. Für multiple Verfolgungen ist aus diesem Grund ein Objekterkennungsschritt während des Trackings nötig, um eventuelle verlassende oder hinzukommende Objekte zu tracken. Wie in Abb.18 zu sehen, werden die zuvor erkannten und identifizierten Objekte einzeln verfolgt, trotz ungleich gerichteter Bewegungen.

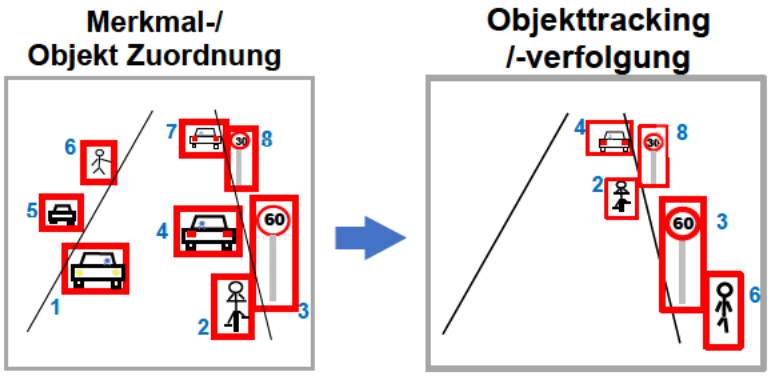


Abbildung 18

Beim Einzelobjekttracking ist das zu erkennende Objekt mit einer Bewegung einhergehend und auf genau eine Möglichkeit für ein zu trackendes Objekt beschränkt. Jedoch ist es möglich mehrere begrenzte Möglichkeiten für Objekte vorab zu definieren und je nach aktueller Aufnahme eines dieser möglichen Objekte zu tracken. Eine scheinbare Lösung für multiple Verfolgung basierend auf einer Einzelobjektverfolgung, wäre eine Reihe von mehreren Unterobjekte, beispielsweise verschiedene Merkmalspunkte einer Aufnahme eines Objekts, die eine gleichgerichtete Bewegung beibehalten, da sie stets ein Teil eines einzelnen realen Objekts sind und somit physisch an die Bewegung gebunden sind. In Abb.19 wird veranschaulicht wie die Unterobjekte der Bewegung des Objekts folgen und sozusagen ein Einzelobjekt in ihrer Bewegung darstellen. Der Vorteil eines derartigen Trackings, statt lediglich eines der Unterobjekte zu tracken, besteht darin mehrerer Punkte auf einem Objekt festzuhalten, sodass die Wahrscheinlichkeit eines scheiternden Trackings des Objekts aufgrund von physischen Umgebungsbedingungen, wie zum Beispiel eine teilweise Verdeckung oder lokal schlechte Lichtverhältnisse minimiert wird.

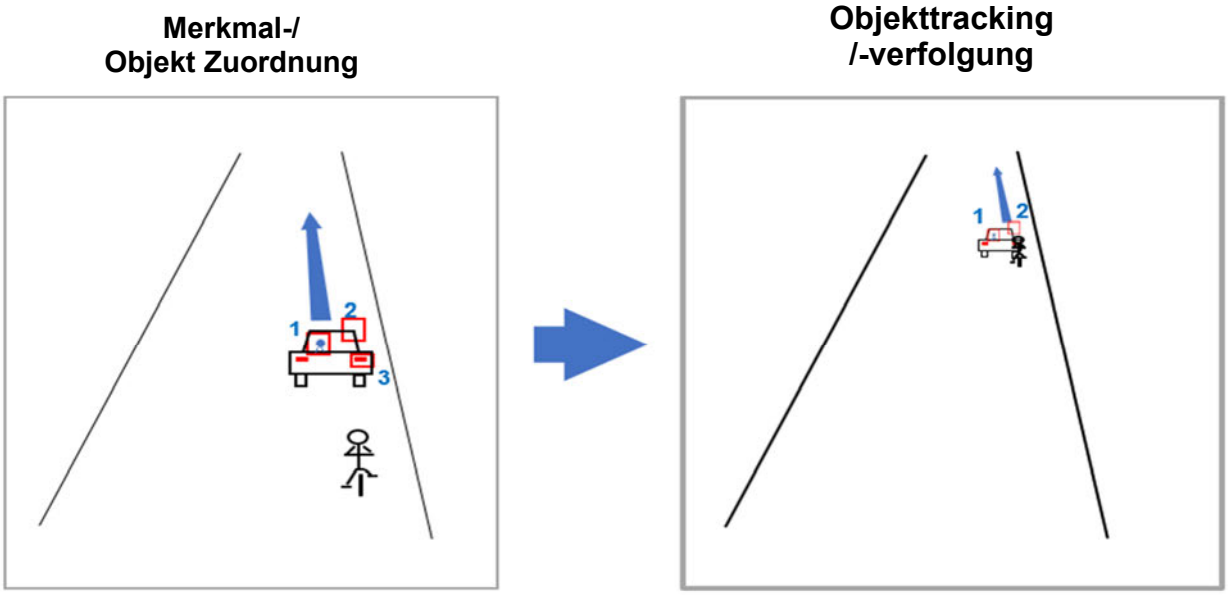


Abbildung 19

Bei Betrachtung der Eigenschaften der Verfolgung von erkannten Merkmalen, stellt sich die Frage, ob ein Detektor, wie zum Beispiel im vorangegangen Unterkapitel präsentiert, nicht als Tracker fungieren kann. Zum Beispiel durch wiederholtes erkennen der Merkmale zu jeder neuen Aufnahme. Der Unterschied und Grund, warum wir nicht sagen können, dass Feature-Detektoren auch gleich Tracker sind, ist ihre Inkonsistenz zwischen den Aufnahmen. Tracker folgen einer Trajektorie, Detektoren übernehme nur die in der momentanen Aufnahme am besten erkennbaren Merkmale, was dazu führt, dass sie sehr instabil sind, besonders, wenn sie das verfolgte Objekt mit etwas verwechseln, das im Moment dem ursprünglich definierten Objekt ähnlicher ist. Daher wird für eine zuverlässige Verfolgung ein speziell darauf ausgelegter Algorithmus benötigt. In den folgenden beiden Unterkapiteln sollen die unterschiedlichen verfügbaren Algorithmen und wie sie in Matlab implementiert zur Verfügung stehen analytisch verglichen werden, um eine Aussage über die jeweilige Eignung, als Auswahl für einen Objekttrackingalgorithmus in einer AR Anwendung, zu treffen.

2.5.2.1 Funktionsweise und Vorstellung des gewählten Objekttracking-Algorithmus

Unter Berücksichtigung der Ausgangslage eines punktbasierten Erkennungsalgorithmus bzw. nach der Erkennung von Merkmalspunkten, ist der KLT-Algorithmus als punktbasierter Trackingalgorithmus am geeignetsten. Grundsätzlich sind auch andersartige, sensorisch basierte Tracker denkbar. In Matlab ist der genannte Tracker, jedoch der einzige punktbasierte Tracker. Abgesehen davon sind alle in 2.4 genannten Trackingmethoden in Matlab einsetzbar. Sie fallen jedoch direkt im qualitativen Vergleich aus der engeren Wahl raus, da sie lediglich Bewegungen ohne spezifische Feature Angabe tracken können. Im nachstehenden Unterkapitel wird auf die Funktionsweise des gewählten Algorithmus eingegangen.

Nach der, in 2.5.2, erläuterten Hauptaufgabe eines Objekttrackingalgorithmus oder auch kurz Trackers gilt es die Verschiebung von bereits erkannten Bildmerkmalen zu finden und deren neue Koordinaten auszugeben. Der Kanade-Lucas-Tomasi Algorithmus, auch kurz KLT Algorithmus genannt, wurde ursprünglich entwickelt, um das Problem zu lösen, dass herkömmliche Erkennungs- und Trackingmethoden im Allgemeinen kostenintensiv werden lässt. Für die Suche nach der besten Übereinstimmung zwischen zwei Bildern nutzt der Ansatz die Intensitätsinformationen. Der Vorteil des Algorithmus gegenüber anderen Lösungen ist die Schnelligkeit bei der Lokalisierung von übereinstimmenden Trackingpunkten, da viel weniger potenzielle Übereinstimmungen zwischen den Bildern gefunden werden, aufgrund einer Auslese von ausschließlich geeigneten Punkten. Das Konzept ist in dem verfassten Artikel, der Entwickler, des Ursprungskonzepts „Detection and Tracking of Point Features“ [31] im Detail nachzulesen. In Anlehnung an den genannten Artikel lässt sich das Konzept wie folgt erläutern:

Die schon erwähnten Intensitätsinformationen werden nach diesem Ansatz mit der Funktion I über drei Variablen beschrieben, wie in *F5*. Dabei sind die Variablen x und y die Raumkoordinaten und gleichzeitig der zu betrachtende Bildpunkt, sowie t der Zeitpunkt eines aktuellen Bildes einer bewegten Aufnahme. Darüber hinaus gilt Zusammenhang *F6*, indem τ das Inkrement und damit die Zeitdifferenz zwischen dem aktuellen und dem vorangegangenen Bild definiert. Ebenso beschreiben ξ und η die Differenz zwischen den Positionen des Punktes in den zu vergleichenden Bildern. Daraus ergeben sich die Verschiebungen und damit die gesamte Verschiebung d, *F7,* des Punktes innerhalb der Zeitintervalls t und $t + \tau$ bzw. dt, wie es vektoriell in Abb. 20 visualisiert wird.

$$I(x, y, t); \quad F5$$

$$I = (x, y, t + \tau) = I(x - \xi, y - \eta, t); \quad F6$$

$$d = (\xi, \eta); \quad F7$$

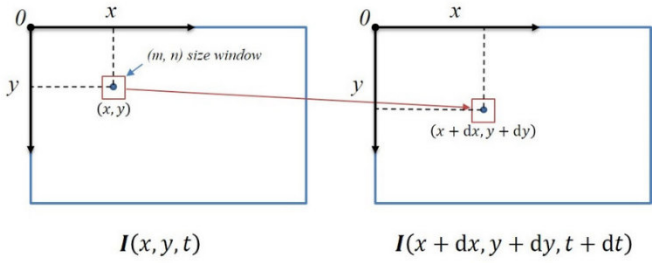


Abbildung 20 [44]

Die Merkmale werden nicht durch den Tracker lokalisiert, sondern werden bereits als initialer Input nach Auslösung des bewegten Bildes übernommen. Die Ermittlung dieser Verschiebung d erfolgt nicht auf der Verfolgung einzelner Pixelpunkte, da dies nahezu unmöglich ist, da sich umliegende Pixel zu sehr in ihrer Helligkeitsintensität ähneln, sondern auf Basis der Pixelbereiche, die zum Beispiel zuvor von einem Merkmalserkennungsalgorithmus erkannt wurden. Insofern ist der Punkt mit den Koordinaten x und y ein Pixelbereich, der ein markantes Muster enthält. Die Verfolgung findet kontinuierlich bei jeder Bildänderung bzw. nach jeder neuen Aufnahme statt. Die neuen Positionen gehen aus der vektoriellen Addition der Ortsvektoren mit den Verschiebungsvektoren.
Einen gewissen Bereich, mit einem enthaltenen Muster zu verfolgen bringt jedoch auch Herausforderungen mit sich. So können beispielsweise dreidimensionale, reale Objekte, aus denen das zweidimensionale Merkmal in der Aufnahme erkannt wurde, durch Veränderung der Oberfläche

des Objekts Schwierigkeiten beim zweidimensionalen Punkttracking verursachen. Im Fall einer Änderung des Musters des Merkmals bei einer entsprechenden Bewegung, innerhalb des eingrenzenden Bereichs, wird dieses verworfen und nicht als Verfolgungspunkt weiterverwendet. Mit dieser ständigen Prüfung und dem Ausschluss werden nur die zuverlässig verfolgbaren Merkmale übernommen. Damit werden auch Trackingfehler auf ein Minimum reduziert.

Ein häufiges Problem taucht jedoch bei Merkmalen auf, die an Objekträndern liegen, die derart gekrümmt sind, dass die zweidimensionale Projektion eines Punktes der Objektoberfläche eine beschleunigte Bewegung an den Rändern widerfährt. An diesen Punkten liegt auch eine sogenannte verschließende Kante vor, „Occluding Edge“ [31], ab der nicht mehr alle Flächen bzw. ein Teil in der zweidimensionalen Ansicht sichtbar sind (s. Abb. 21).

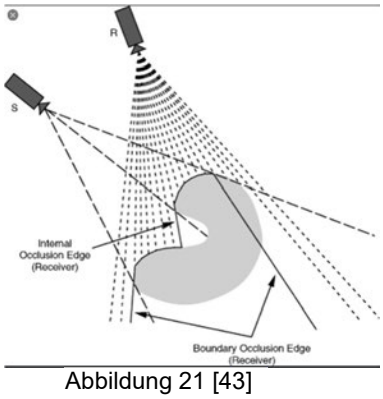


Abbildung 21 [43]

Jedoch gibt es einen Ansatz für eine Lösung des Problems die in [31] benannt wird. In Abhängigkeit des Verschiebungsvektors d wird überprüft, ob mit diesem jegliche Diskrepanz zwischen den Aufnahmen beschrieben werden kann. Wenn dies nicht der Fall ist, gilt die Pixeldarstellung des Merkmals als Fehler und der Verschiebungsvektor wird dahingehend korrigiert. Damit findet in der Regel eine insgesamte Schätzung der Verschiebung für jede Iteration statt. Mit der endgültigen Verschiebung werden die resultierenden Werte für die Intensitätsfunktion ausgegeben und somit die neuen Merkmalskoordinaten festgelegt. Darauf basierend wird eine Transformation in der projizierten Ebene möglich.

2.6 Synthese und Analyse von Lösungsmöglichkeiten für eine Augmented-Reality Umgebung

Für die Umsetzung einer einfachen Augmented-Reality Umgebung in Matlab gibt es verschiedenste Lösungsmöglichkeiten unter Verwendung der vorgestellten notwendige Bestandteile und Algorithmen. In diesem Abschnitt 2.6 werden drei Konzepte, resultierend aus verschiedenen Lösungsideen, behandelt und präsentiert. Zugleich wird hiermit die zweite Phase der Entwicklung dargelegt. Es handelt sich bei den Konzeptvarianten um allgemeine algorithmische Ansätze für eine Prozedur während der Laufzeit einer bewegten Aufnahme. Für **alle Konzepte** werden die aus der vorangegangenen Untersuchung **gewählten Algorithmen der Objekterkennung und des Objekttracking** genutzt. Die Varianten sind an die Lastenheft Anforderungen und der Zielsetzung schematisch in Komponenten unterteilt und unterscheiden sich hauptsächlich in ihrer Umsetzung der funktionalen Anforderungen, da sie alle mindestens die notwendigen Bestandteile abdecken. Auf die Lösung der Aufnahme von Bildern für die Erzeugung eines bewegten Bildes wird an dieser Stelle nicht eingegangen, da in Kapitel 3 die Umsetzung erläutert wird. Somit werden in den Konzepten die Lösungen für die funktionalen Anforderungen **Erkennung**, **initiale Position**, **Tracking**, **Berechnung der neuen Positionierung**, **CAD Datenzugriff**, **Echtzeit Überlagerung** gefunden und dargelegt.

2.6.1 Lösungsideen und ihre Eigenschaften

Erkennung

Gegenüber stehen die **markerlos- und markerbasierten** Varianten:

Ebenen - Erkennung

Der Objekterkennungsalgorithmus wird derart eingesetzt und parametrisiert, dass keine Marker Referenz für die Initiierung der weiteren Prozedurschritte erforderlich ist. Es soll die abgebildete Umgebung in der Momentaufnahme auf alle erkennbaren Merkmale durch die Erkennungsfunktion untersucht werden. Statt einer ständigen Wiedererkennung von Marker-Merkmalen entsteht hier bei jeder Laufzeit, bedingt durch die Umgebung, ein beliebiges Muster. Es entsteht sozusagen ein einmaliger, individueller Marker als Verankerung, im Moment der ersten Aufnahme, der als fixierte Orientierung und Identifikation dienen soll. Die Bedingung für die Weiterleitung, an den nächsten Prozedurschritt, ist das **Erkennen des beliebigen Musters in einer Ebene**, sodass diese zur Orientierung und Ausrichtung für ein globales Koordinatensystem dient.

Marker - Erkennung

Für diese Variante soll eine markerbasierte Erkennung stattfinden. Wie bereits in 2.4 erläutert, kommt ein Referenzgrafik zum Einsatz. Der Marker wird in der realen Welt, zum Beispiel als ausgedruckte Fotokopie, positioniert. Die räumliche Erkennung findet durch das Mapping der erkannten Muster der Umgebung mit dem hinterlegten Referenzbild statt. Die Anwendung wird gestartet und die Umgebung als bewegte Aufnahme dargestellt und mit dem Objekterkennungsalgorithmus analysiert. Für jedes Bild findet eine Untersuchung durch den Algorithmus statt. Sobald der Marker in der aufgenommenen Umgebung gefunden wurde, wird der nächste Prozedurschritt initiiert und die Bildkoordinaten an die nächste Verarbeitung übergeben und die Erkennung vorübergehend beendet.

Initiale Position

Bildmittelpunkt - initiale Position

Die erkannten Umgebungsmerkmale werden danach für die **initiale Position**, aufgrund maximaler Anzahl, gefundener Merkmalspunkte, verwendet. In Relation zu diesen Punkten wird in der Mitte des betreffenden Bildes, der Ursprung des globalen Koordinatensystems festgelegt. Dabei wird aufgrund der variierenden Oberflächen der Realität, bei markerloser Erkennung, die Ausrichtung der Koordinatenachsen standardmäßig so eingestellt, dass das die x-y Ebene planar mit der Ebene ist, die von einem beliebigen Muster aus Merkmalspunkten abstammt.

Markermittelpunkt - initiale Position

Nach der Übergabe der Bildkoordinaten des Markers, auf dem aufgezeichneten Bild, werden diese von der initialen Positionierung übernommen und verarbeitet. Mit Hilfe der gefundenen Merkmale aus der Erkennung, die sich in jedem Fall auf dem Marker befinden, kann ein gemittelter Mittelpunkt des Markers in globalen Koordinaten ermittelt werden. Die Koordinaten der

verfügbaren Merkmalspunkte werden angenähert und zu einem Punkt zusammengefasst. Dieser singuläre Punkt dient zur Positionierung des Ursprungs des globalen Koordinatensystems.

Tracking

Kinematisch visuelles - Tracking

Die Verfolgung der Bewegungen, findet auf visueller Basis mit Hilfe des Objekttrackingalgorithmus statt und verfolgt alle zuvor erkannten Merkmale, um die Verschiebungsvektoren für das nächste Bild der Bewegung zu erhalten. Neben des visuellen Trackings werden hier auch die Bewegungsänderung, die aus der Datenerhebung eines Beschleunigungssensors hervorgehen, berücksichtigt, um die Änderungsrate der Bewegungsrichtung zu prognostizieren.

Visuelles - Tracking

Das Objekt wird ausschließlich visuell getrackt und dafür erkannte Merkmale des Markers vom Beginn der Prozedur verwendet. Es findet eine Validierung statt, um nur die treffendsten Merkmalspunkte des Markers zu verwenden. Diese gelten ab diesem Punkt als fixierte Trackingpunkte und werden verfolgt. Aus ihren sich ändernden Koordinaten wird die gesamte Bewegung der Kameraperspektive im Raum hergeleitet. Als Ergebnis der ständigen Verfügbarkeit von neuen Koordinaten der Trackingpunkte, werden diese an die Berechnung der neuen Position übergeben.

Berechnung der neuen Positionierung

Gyrometerbasierte visuelle - Berechnung der neuen Positionierung

Da, wie schon beschrieben, zwischen den Umgebungsmerkmalen und der initialen Position ein proportionaler Zusammenhang besteht, erfolgt in dieser Variante eine Transformation der Verschiebungsvektoren der getrackten Punkte, die sich außerhalb des Objektkoordinatenursprungs befinden, in Objektkoordinaten. Damit werden die translatorischen Positionsänderungen berücksichtigt. Die Rotation der Kameraansicht und damit auch die Änderung der relativen Koordinatenachsenausrichtung zur Ausgangslage und Blickwinkel, werden durch kinematische Sensoren ermittelt. Dabei greift man auf den Gyrosensor des Endgeräts in er Laufzeitumgebung zu.

Visuelle - Berechnung der neuen Positionierung

Für die Translation werden die Richtung und der Betrag der Vektoren genutzt. Für die Rotation des Blickwinkels auf das Objekt wird eine Transformation, einer endlichen projektiven Ebene, die durch mindestens 4 Punkte aus den zur Verfügung stehenden Trackingpunkten definiert wird, genutzt um die Winkeländerungen der Geraden als beeinflussende Variablen für Ausrichtung der Koordinatenachsen in Relation zur aktuellen Kameraperspektive. Als mathematischer Ansatz kann die Thematik aus „projective planes“ von Marshall Hall aufgegriffen werden [32].

CAD Datenzugriff

Cloud - CAD Datenzugriff

Der Zugriff auf die benötigten 3D **CAD Daten** erfolgt über einen zentralen, geräteunabhängigen Datenspeicher. Denkbar ist für diese Aufgabe eine Cloud Lösung. Es bietet sich konkret in diesem Fall die „Matlab Drive Cloud“ an [33]. Dies ist eine Dienstleistung, die von Mathworks in Kombination mit Matlab angeboten wird, um einen schnellen Datenaustausch zwischen mehreren Geräten zu vollziehen auf den Matlab jederzeit Zugriff hat. Denkbar ist im Allgemeinen ein Datenaustausch zwischen dem ausführenden Endgerät und einem zentralen Server, der die CAD Daten speichert.

Lokaler – CAD Datenzugriff

Für die Erhebung der CAD Daten soll ein Zugriff auf einen lokalen Datenspeicher erfolgen. Dieser setzt eine Mindest-Speicherkapazität des darzustellenden CAD Objekts als STL Datei voraus. Idealerweise liegt ein lokales Verzeichnis mit genügend Kapazität für mehrere abrufbare STL Dateien zur Verfügung. Der Austausch von neuen Daten oder Updates von CAD Modelldaten, muss über eine Dateiübertragung der CAD Daten auf das Gerät, mit der Matlab Laufzeit Umgebung, stattfinden.

Echtzeit Überlagerung

Patch STL File Reader - Echtzeit Überlagerung

Die Darstellung der CAD Daten wird durch ein Rendering aus Patches, mittels einer Funktion, die bereits in 2.4 vorgestellt wurde, realisiert. Der geometrische Mittelpunkt des Objekts wird dabei auf den globalen Ursprung des Koordinatensystems, der sich bei unmittelbarer Sicht auf die Ausgangsposition im Sichtfeld befindet. Mit der „patch“ Funktion in Matlab können die Daten, die durch den „STL File Reader“ generiert werden, gerendert und somit optisch dargestellt.

In der folgenden Tabelle (Abb.21) wird veranschaulicht, welche Lösungsideen, in welcher Kombination für eine Gesamtlösung und damit als Grundlage für ein Konzept in Frage kommen und anschließend zur Bewertung stehen. Es werden drei Varianten dargestellt.

Funktionale Anforderung	Lösung A	Lösung B	Lösung C
Erkennung	Ebenen	Marker	Marker
Initiale Position	Bildmittelpunkt	Markermittelpunkt	Markermittelpunkt
Tracking	Kinematisch-visuell	Visuell	Kinematisch-visuell
Berechnung der neuen Positionierung	Gyrometerbasiert-visuell	Visuell	Visuell
CAD Datenzugriff	Lokal	Lokal	Cloud
Echtzeit Überlagerung	Patch	Patch	Patch

Abbildung 22

2.7 Bewertung und Konzeptfestlegung

In der untenstehenden Grafik (Abb. 22) ist die Bewertungsmatrix für die Lösungsvarianten A bis C abgebildet. Die Bewertungskriterien werden dabei mit einer Wertung von 1 bis maximal 3 Punkten bewertet. Dabei ist die Bewertung mit höherer Punktzahl aufsteigend und entspricht einer Rangfolge für die am besten vorliegende Lösungsidee der jeweiligen Kategorie. Die Punkte werden in jeder Kategorie separat gewichtet. Die Punktzahlen der einzelnen Kriterien werden in drei rechten Spalten addiert und der Mittelwert über diesen gebildet.

Gew.	Bewertungskriterien	A	B	C	**A**	**B**	**C**
1	Hardwareanforderung	1	3	2	1	3	2
1,25	Kompatibilität	1	3	2	1,25	3,75	2,5
1	Umsetzbarkeit	1	3	1	1	3	1
1	Erweiterbarkeit	3	1	2	3	1	2
1,5	Erkennung	3	2	2	4,5	3	3
1	Initiale Position	2	3	3	2	3	3
1,5	Tracking	3	2	3	4,5	3	4,5
1	Ber. der neuen Position.	3	2	2	3	2	2
1	CAD Datenzugriff	2	2	3	2	2	3
1	Echtzeit Überlagerung	1	1	1	1	1	1
∑	**Gesamtbewertung**				**2,325**	**2,475**	**2,4**

Abbildung 23

Die einzelnen Bewertungen werden, in Anbetracht dieser Arbeit, basierend auf folgenden Hypothesen getroffen:

Beginnend mit den Hardwareanforderungen ergibt sich als beste Lösung die Variante B, da die beste Lösung die niedrigsten Hardwareanforderungen beansprucht. Dies lässt sich damit begründen, dass für B nur ein optischer Sensor gebraucht wird, der mindestens in der Lage sein muss, Bilder der zu erweiternden Umgebung aufzuzeichnen. Für A wird die niedrigste Bewertung gewählt, weil hier, neben der Kamera, ein Beschleunigungs- und ein Gyrosensor, benötigt werden. Insgesamt werden bei A drei Sensoren eingesetzt, wohingegen C zusätzlich nur den Beschleunigungssensor und somit nur zwei Sensoren verwendet.

Die Kompatibilität wird bei B am stärksten bewertet, da mit der Verwendung rein visueller Erkennungs- und Trackingalgorithmen, ein vereinfachter Abruf der Sensordaten ermöglicht wird und lediglich ein Kanal notwendig ist. Daher ist diese Variante mit den meisten Geräten und Laufzeitumgebungen kompatibel, falls die Anwendung in eine andere gängige Programmiersprache konvertiert werden muss, um diese zu implementieren.

Bei der Umsetzung von B ist die Umsetzbarkeit am ehesten gegeben, weil die markerbasierte Erkennung eindeutiger und mit weniger Aufwand zu identifizieren ist. Für die Ebenen Erkennung, in A, ist eine Analyse der gefundenen Punkte notwendigen, die in der Lage ist die dreidimensionale Position der Punkte aus einer zweidimensionalen Aufnahme heraus zu interpretieren,

sodass nur die Punkte übernommen werden, die planar zueinander liegen und eine Ebene aufspannen. Darüber hinaus unterscheidet sich die Variante C besonders im Tracking Ansatz von B, wobei dieser Ansatz eine Erweiterung um die kinematische Optimierung eines visuellen Trackings darstellt und daher zusätzlich zur Schleife des rein visuellen Trackings, in B, eine gleichzeitige Verarbeitung der Beschleunigungsdaten innerhalb derselben Iteration voraussetzt.
Die Erweiterbarkeit von A und C sind fast gleichauf, wenn man berücksichtigt, dass drei Kanäle durch die Anzahl der Sensoren verwendet werden. In A entfällt jedoch die Marker Erkennung und daher eine größere Vielfalt der Erweiterungen möglich, weil der Prozess nicht an eine physische Limitierung durch den realen Marker, wie zum Beispiel die Skalierung gebunden ist.
Für den markerlosen Ansatz der Erkennung in A wird die höchste Bewertung gewählt, da sie die handlichste Lösung ohne die Notwendigkeit von physischen Markern darstellt. Variante B und C werden gleich bewertet, da die Erkennung in beiden Fällen dem markerbasierten Ansatz folgen.
Die initiale Position wird bei B und C gleich und damit stärker als bei A bewertet. Die initiale Position ist bei etwas aufwendiger zu ermitteln. Es wird bei A zwar nur der aktuelle Bildmittelpunkt übernommen, jedoch ist eine variierende Positionierung in dieser Methode, die Position von der momentanen Ausrichtung der Kamera und der zugrundeliegenden verfügbaren realen Umgebung. Der Vorteil beim Marker Mittelpunkt zur initialen Positionierung ist die stets gleiche Positionierung, die in Abhängigkeit des physischen Markers verlässlicher und nach Belieben wählbar ist.
Das kinematisch-visuelle Tracking in Variante A und C wird am höchsten bewertet, da die Unterstützung durch den Beschleunigungssensor, im Gegensatz zur rein visuellen Methode, eine Approximation zum Verlauf der folgenden Bewegung ermöglicht. Der große Vorteil besteht darin, bei einer unzureichenden (im weiteren Sinne eine zu schnelle Bewegung für das optische Tracking bei gleicher Bildrate) Verfolgung der Punkte, dass die Bewegung ohne visuelle Eindrücke verfolgt werden kann. Der Aufwand steht der rein visuellen Methode im Nachteil gegenüber. Aus diesem Grund wird dieser Aspekt bei Variante B trotzdem mit 2 Punkten versehen.
In allen Varianten arbeitet die Berechnung der neuen Position im Allgemeinen auf der gleichen Basis, dem Transformieren der Koordinaten basierend auf den Verschiebungsvektoren des Trackings. Jedoch bietet die Erweiterung der Winkelbestimmung durch einen Gyrosensor in Variante A einen Vorteil bei der Zuverlässigkeit der Ausrichtungsbestimmung, da sie von den optischen Gegebenheiten entkoppelt ist.
Die Variante C wird aufgrund des CAD Datenzugriffes über eine Cloud am höchsten gegenüber den anderen beiden Varianten gewertet, da die Anwendung hier auf beliebig viele Endgeräte ausgeweitet und der Austausch von Updates zu vereinfacht werden kann. Damit steigt die Dynamik bei der Entwicklung und fördert kürzere Optimierungsschleifen. Außerdem ist mit dem ausgelagerten zentralen Speicherort eine theoretisch jederzeit erweiterbare, geräteunabhängige, Speichergröße erzielbar. Obwohl die Vorteile überwiegen, sollte erwähnt werden, dass die Lesegeschwindigkeit der Daten teilweise eingeschränkt sein kann und aufgrund dessen ein vorzeitiger Download der Daten auf den lokalen Speicherplatz, noch vor der Laufzeit, nötig ist.

Die Darstellung der CAD Daten unterscheidet sich in allen Varianten nicht, da mit dieser Methode bereits eine zufriedenstellende Lösung für das Generieren eines Inputs für die patch-Funktion in Matlab gewährleistet ist. Unabhängig von der Laufzeitumgebung wäre die STL File Reader Funktion adaptierbar, da die **CAD Daten auf binärer Ebene ausgelesen** werden. Die anschließende Verarbeitung der Daten mit der patch-Funktion stellt eine einfach zu implementierende Methode dar, die auch in anderen Entwicklungsumgebungen analog realisiert werden kann.
Die **Festlegung auf ein konkretes Konzept** fällt auf die Wahl von **Variante B**, mit dem markerbasierten, rein visuellen, lokalen Ansatz. Nachfolgen soll das auszuarbeitende Konzept noch einmal im Detail modelliert und veranschaulicht werden.
Zunächst soll eine Gesamtarchitektur mit allen Komponenten des Augmented Reality System feststehen. In der untenstehenden Abb.24 sind die übergeordneten Komponenten des konkreten Systems des Konzepts abgebildet. Zu sehen ist, die Hardware, Software und der Server. Als Software Teilkomponente ist Matlab als Laufzeitumgebung verfügbar und hat Zugriff auf die Teilhardware Kamera und Datenspeicher. Auf dem Datenspeicher befinden sich Objektdaten, für die optische Überlagerung dieser durch die Anwendung, ein Referenzbild des zu verwendenden Markers und das Skript der Anwendung selbst.

Komponentendiagramm

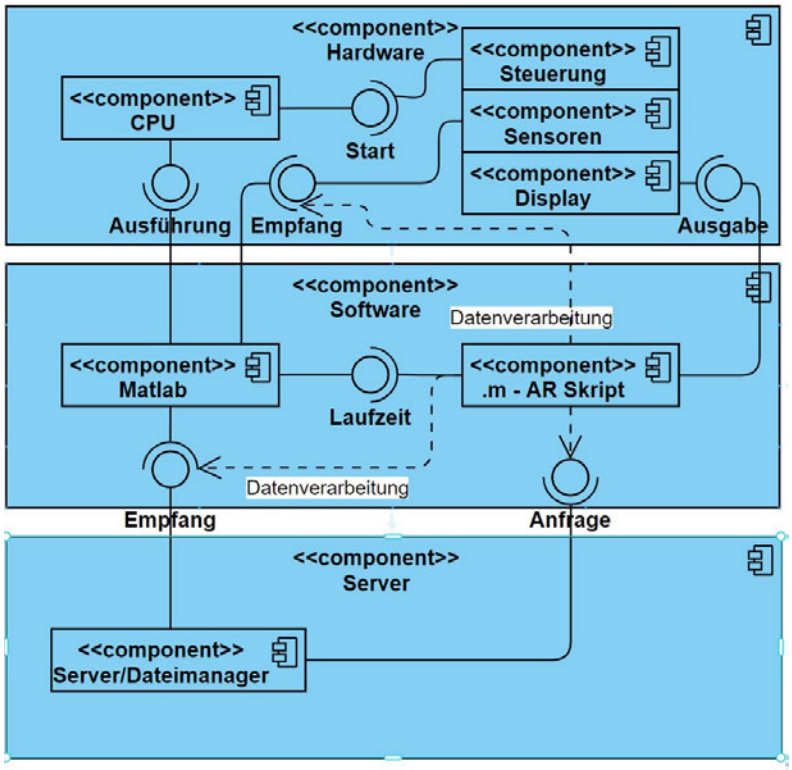


Abbildung 24

In der darauffolgenden Abb. 24 wird das Konzept für den avisierten Programmablaufplan nach DIN 66001 als Flussdiagramm präsentiert. An dieser Stelle soll eine simplifizierte Darstellung erfolgen, um den Ablauf des zu entwickelnden Skripts für die Anwendung reduziert in seinen wesentlichen Elementen zu beschreiben.

Programm Ablaufplan:

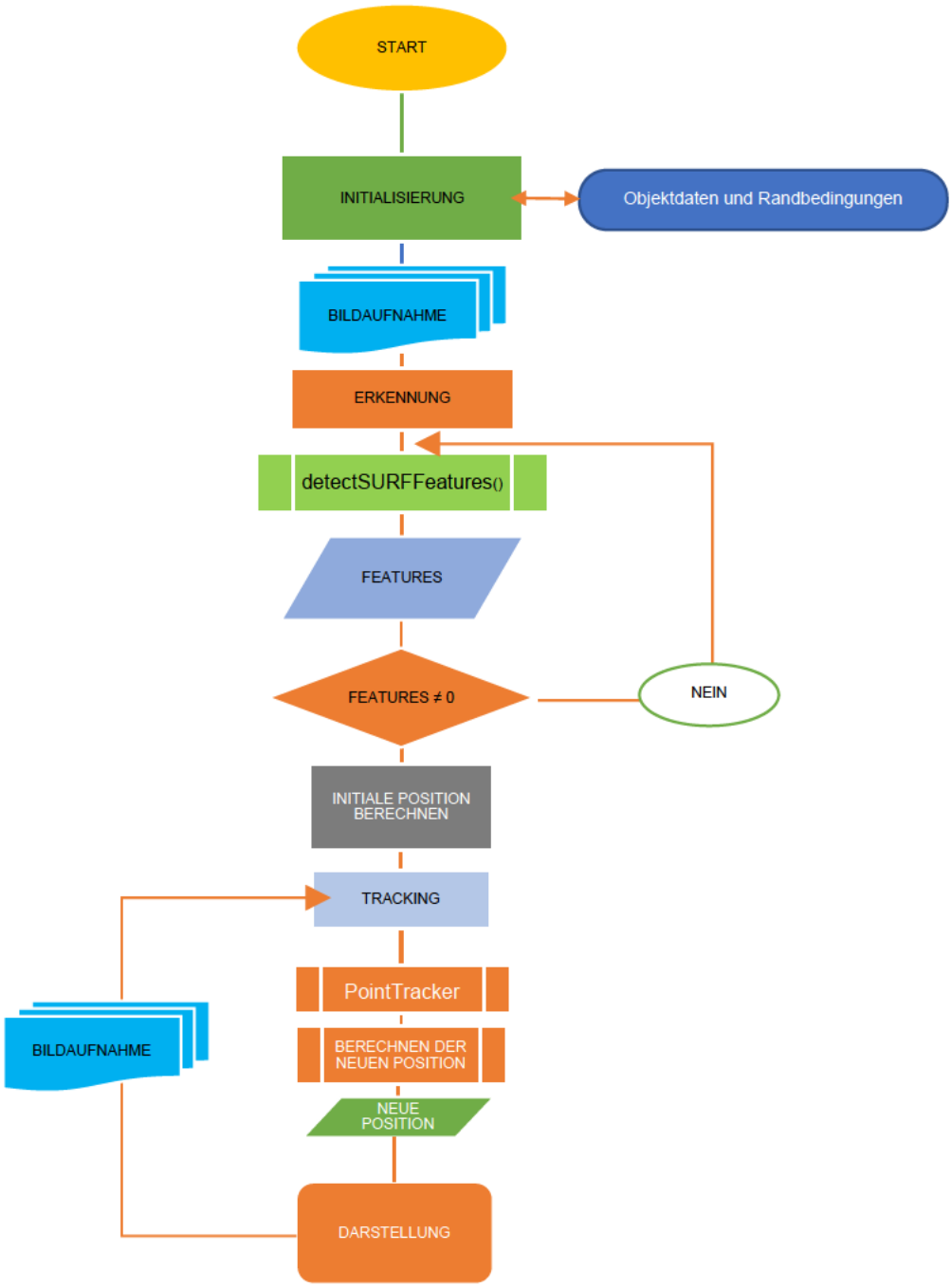


Abbildung 25

3 Umsetzung und Vorstellung der Augmented-Reality Umgebung in Matlab

3.1 Vorbereitungen und Annahmen für die Umsetzung auf einem mobilen Endgerät

Vor Beginn der Umsetzung des konkreten Konzepts müssen Annahmen bestätigt oder aufgeklärt werden, um die Kompatibilität des mobilen Endgeräts, in diesem Fall ein Smartphone, als Laufzeitumgebung zu gewährleisten.
Als eines der wichtigsten Bestandteile gilt es zu klären wie eine bewegte Aufnahme in Matlab verarbeitet werden kann. Es muss also sichergestellt sein, dass die Bilddaten übergeben werden können. Eine Verarbeitung der bewegten Aufnahme muss für eine Überlagerung in Echtzeit, während die Aufnahme erfolgt, geschehen.

Darüber hinaus sind gewisse Vorbereitungen notwendig, um die Anwendung auf einem handelsüblichen Smartphone zu betreiben:
Für die anschließende Implementierung auf ein handelsübliches mobiles Gerät ist ein Smartphone mit mindestens einer iOS 11 oder Android 6.0 Version als Betriebssystem für die Matlab Mobile App [34] notwendig. Außerdem ist eine integrierte Kamera, über die man mittels der Matlab Mobile App zugreifen kann, eine Hardwarevoraussetzung. Im Einklang mit diesen Voraussetzungen wird ohne konkrete Ermittlung der Systemanforderungen angenommen, dass das ausführende Smartphone eine über eine ausreichende CPU verfügt, um der Anwendung gerecht zu werden. Somit fokussiert man sich hier lediglich auf die Anwendung als solche und nicht auf die Kompatibilität auf allen Systemen.
Die Prototyp Versionen werden vorerst als Desktop Version entwickelt. Ziel ist es somit im ersten Schritt einen Prototyp auf einem Notebook oder PC mit angeschlossener Kamera, mit der Desktop Version von Matlab, erfolgreich laufen zu lassen. Die Implementierung in eine Smartphone Version erfolgt danach.

3.2 Erstellung einer einfachen Augmented-Reality Umgebung für die optische Überlagerung von 2D-Objekten

Als ersten Schritt bei der Realisierung einer prototypischen Lösung wird eine zweidimensionale Lösung angestrebt. Ziel ist es, bei dieser Anwendung, die Anforderungen mit der optischen Überlagerung eines Bildes auf dem zu verwendenden Marker umzusetzen.
Das aus der Erörterung, in 2.7, hervorgehende Konzept wird übernommen und die Logik der einzelnen Programmablaufschritte wird stückweise als konkrete programmierbare Lösung entwickelt.

3.2.1 Initialisierung der Randbedingungen und Objekte

Einleitend muss die notwendige Initialisierung der Randbedingungen und notwendigen Objekte erfolgen. Wie im Folgenden zu sehen wird das Referenzbild des Markers „refimg“ (Abb. 26) eingelesen. Darauffolgend findet eine Konvertierung in ein Graustufenbild statt, da man für die

„detectSURFFeatures“ Funktion lediglich eine Auswertung der Kontrastverhältnisse erzielt und ein Graustufenbild als Input ausreicht. In „replacemedia“ (Abb. 26) wird das Austauschmedium gespeichert, dass in späterer Berechnung mit der Aufnahme des Markers optisch überlagert werden soll und die Realität mit dessen Ansicht erweitert. Sowohl das Referenzbild als auch das Austauschbild werden als vorzeichenlose 8-Bit integer Arrays (*auch: unsigned 8-Bit integer Array, kurz: uint8*) gespeichert, mit Werten von 0 bis 255. Die Kamera Verbindung wird mittels der Funktion „webcam“ auf der Desktopversion erzeugt und initialisiert, wobei diese die als erste auffindbare optische Hardware als Kamera definiert (meistens gibt es nur eine eindeutige Kamera bzw. Webcam am Gerät, sodass in der Regel bei einem üblichen PC/Laptop keine spezielle Definition notwendig ist; Beim Smartphone wiederum wird eine Unterscheidung notwendig Front- bzw. Rückkamera).
Die genannte Funktion ist in der „Image Acquisition Toolbox“ erhältlich und stellt eine der schnellsten und handlichsten Methoden für das Erlangen von Hardware-Bilddaten in Matlab dar [35]. Die erhältliche Funktion erzeugt eine eigenständige Klasse, die ein Objekt namens „webcam“ erzeugt, dass ein konstantes Streaming der Bildaufnahmen zu Matlab aufrechterhält. Somit kann jederzeit aus dem Arbeitsspeicher mit Matlab ein Bild ausgelesen werden und mit dem „webcam“-Objekt die abrufbaren Geräteeigenschaften, wie die Auflösung, Bildschärfe, Bildrate und weitere manipuliert werden. Standardmäßig wird das Objekt mit den höchsten verfügbaren Eigenschaften der jeweiligen Kategorie, wie beispielsweise der maximalen Auflösung generiert, sodass eine zusätzliche Eigenschaftsanpassung nicht anfällt.

```
% initi. Referenzbild 'refimg' und Bildmerkmale erfassen
  refimg = imread('Marker_Referenz.jpg');
  %erkennen + extrahieren der 'SURF Features'
  refimggray = rgb2gray(refimg);
 %init. Austauschmedium 'replacemedia'
   replacemedia = imread('sample.jpg');

 %init. Kamera 'cam' und Bildmerkmale erfassen
   %verbindung zur Kamera aufbauen
   cam = webcam;
```

Abbildung 26

3.2.2 Erste Schritte zur Realisierung eines Mappings zwischen einem realen Marker und dessen Referenz

Wie schon aus vorangegangen Kapiteln bekannt, ist die Erkennung ein zentraler und ausschlaggebender Bestandteil der Anwendung. Die Genauigkeit und Stabilität hängen maßgeblich von der Erkennung ab. Ebenso relevant ist nach der erfolgreichen Erkennung von Mustern, diese auch zu interpretieren und mit dem Referenzmuster bzw. dem digitalen Bild des Markers zu mappen. Wie in Abb. 28 zu sehen soll jedes Feature, das in der realen Aufnahme des Markers gefunden wird, einem Feature der digitalen Version des Markers zugeordnet und somit erkannt werden, dass es sich bei dem erkannten Muster um den initiierenden Marker handelt.

Daher wird in diesem Schritt versucht die Merkmale beider Grafiken zu erkennen und die Übereinstimmung von Merkmalen mit folgendem Abschnitt (zu sehen in Abb. 27) zuzüglich der vorangegangenen Initialisierung zu lösen und zu testen:

```
% erkennen + extrahieren der Referenzbild - 'SURF Features'
refpts = detectSURFFeatures(refimggray);
refFeatures = extractFeatures(refimggray, refpts);
% Darstellung der 'SURF Features'
figure;
imshow(refimg),hold on;
plot(refpts.selectStrongest(100));

% erkennen + extrahieren der Kamerabild - 'SURF Features'
for a=1:10
      currentframe = snapshot(cam);
      currentframegray = rgb2gray(currentframe);
      campts = detectSURFFeatures(currentframegray);
      % darstellung der akt. Aufnahme
      imshow(currentframe);
      hold on;
      [camFeatures, validpts] = extractFeatures(currentframegray,
campts);
      plot(validpts.selectStrongest(100));
end

idxpairs = ...
matchFeatures(camFeatures, refFeatures,'MaxRatio',0.9);%,...
%'Method','Approximate','MatchThreshold',15);

% speichern der jwl. matched surf pkt.
matchedcampts = campts(idxpairs(:,1));
matchedrefpts = refpts(idxpairs(:,2));
figure;
showMatchedFeatures(...
      currentframe, refimg, matchedcampts, matchedrefpts,'Montage');
```

Abbildung 27

Die „detectSURFFeatures“ Funktion wird hier zweimal angewendet. Einmal auf das Referenzbild, sozusagen das Originalbild des Markers als Bilddatei, und zum zweiten die Kameraaufnahme des Markers als Abbild der physischen Version. Zuerst werden die Referenzmerkmale erkannt. Das zuvor in Graustufen konvertierte Bild wird in „detectSURFFeatures“ eingesetzt. Der Output wird in „refpts“ gespeichert und stellt ein SURFPoints Objekt dar (wie in 2.5 vorgestellt). Anschließend wird das Objekt mit der Funktion „extractFeatures“ bearbeitet um, die Features aus dem SURFPoints Objekt zu extrahieren. Diese werden für die genaue Beschreibung der Merkmale und deren Position auf dem Bild benötigt. Die Features werden in „camFeatures“ gespeichert.

Die darauffolgende Erkennung und Extraktion der erkannten Merkmale der Umgebung wird folgendermaßen umgesetzt:

Im Wesentlichen findet der Prozess analog zu der vorangegangenen Methode statt. In „currentframe“ wird mit der Funktion „snapshot()“ mit dem Input „cam“, als zu verwendendes Kameraobjekt, eine Aufnahme des aktuellen Bildstreamings als uint8 Variable gespeichert. Diese aufgezeichneten Kamera-Bilddaten werden als Graustufenbild konvertiert und in „currentframegray“ gespeichert. Diese Variable wird in die „detectSURFFeatures“ Funktion eingesetzt und dessen Output in „campts“ gespeichert. Im Anschluss wird ebenfalls eine Extraktion der Features vollzogen und hier in „camFeatures“ gespeichert. Dieser Vorgang wiederholt sich in einer zählergesteuerten Schleife (10 Iterationen), in der die Erkennung zu jeder Iteration wiederholt wird. Diese Schleife dient lediglich der besseren Aufzeichnung in der Praxis. Innerhalb dieser Schleife lässt sich einfacher eine brauchbare Aufnahme realisieren. Eine sofortige Aufnahme und Erkennung nach der Initialisierung der Kamera würde zu schnell für eine statische Momentaufnahme der Umgebung sein, falls die Kamera nicht bereits vorher stationiert worden ist. Daher hätte es ein unscharfes Bild zur Folge, das anschließend fehlerhaft durch die „detectSURFFeatures“-Funktion analysiert wird. Während die Schleife läuft sollen die aktuell erkennbaren Features in Echtzeit für jede Iteration sichtbar sein. („imshow()“ und „plot()“). So entsteht die Möglichkeit minimale Änderungen der Ansicht vorzunehmen, um Merkmale erkennbarer zu machen, ohne das Skript vorzeitig neustarten zu müssen.

Abbildung 28

Nach Beendigung der Schleife werden die Features der Referenzgrafik „refFeatures“ und die Features der Momentaufnahme „camFeatures“ in eine Funktion namens „matchFeatures()“ eingesetzt, die seitens Matlab vorgefertigt ist, um die übereinstimmenden Features, laut ihrer Deskription, zu ermitteln [36]. Der Output wird in „idxpairs“ als zweidimensionales boolesches Array gespeichert, um in „matchedcampts“ und „matchedrefpts“ mit den jeweiligen Indexierungen der Grundarrays von „campts“ und „refpts“ zugeordnet zu werden. Mit anderen Worten werden an dieser Stelle nur die Array Werte, an deren Index der Wert von „idxpairs“ mit selbigem Index, wahr bzw. 1 ist, übernommen.

Zu Testzwecken werden die übereinstimmenden, „matching“, Punkte mit der Funktion „showMatchedFeatures“ dargestellt (Abb. 29).

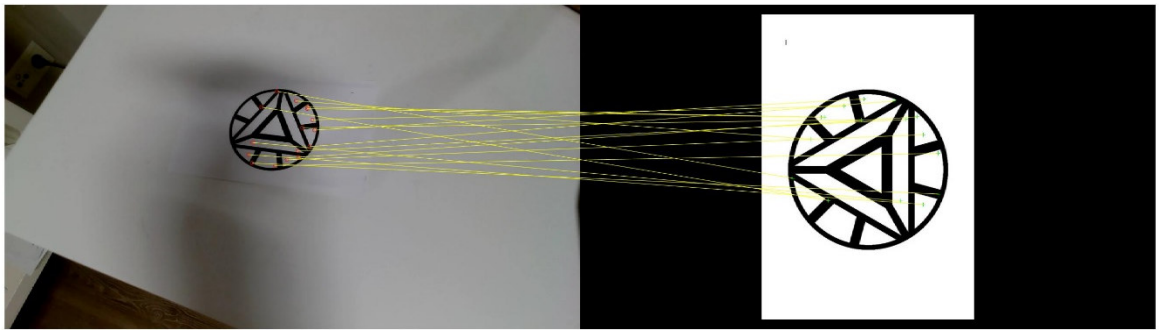


Abbildung 29

3.2.3 Initiale Position: 2D

Nach dem eine Erkennung inklusive des Matchings von Features erfolgreich ist kann die initiale Position festgelegt werden. Für diesen Schritt wird zunächst die Transformation der Marker-/Referenzbildkoordinaten zu einer perspektivischen Ansicht, die der des Markers in der realen

```
%% Transformation zwischen den Referenzbilddaten und den vorliegenden
% Bilddaten bestimmen
    [refTransform, inlierindx] = ...
        estimateGeometricTransform2D(...
                matchedrefpts, matchedcampts,'projective');%'similarity');
    inlierCameraPts = matchedcampts(inlierindx,:);
    inlierReferencePts = matchedrefpts(inlierindx,:);
% in der einflussreichen Vektoren Menge liegende Punktpaare
figure;
showMatchedFeatures(...
     currentframe, refimg, inlierCameraPts, inlierReferencePts,'Montage');
%% Redimensionierung der ersetzenden Mediendaten
% lade Austauschmedium
frame = replacemedia;
% ermitteln der Array Größe: Austauschbild und Marker Referenzbild
repdims = size(frame(:,:,1));
refdims = size(refimg);
%Transformationskalierung ermitteln die das Austauschmedium auf die
%aufgenommene Bildgröße/Markergröße skaliert
scaletransform = estimateScale(refdims,repdims);
outputview = imref2d(size(refimg));
vidframescaled = imwarp(frame, scaletransform,'OutputView', outputview);
%Darstellung des Originalbildes und des transformierten Bildes
figure;
imshowpair(refimg, vidframescaled,'Montage');
%% Transformation auf die ersetzende Mediendaten anwenden
outputview =imref2d(size(currentframe));
vidframetransformed = imwarp(vidframescaled,refTransform, ...
                             'OutputView',outputview);
figure;
imshowpair(currentframe, vidframetransformed,'Montage');
%================================================================================
%% Skalierung fuer die Bildtransformation ermitteln
    function [scaleTransform ]=estimateScale(refDims,repDims)
        if (repDims(1)/repDims(2) > 1) && (refDims(2)/refDims(2) > 1)
            % Wenn das Verhaeltnis zwischen Laenge und Breite gleich ist
            scaleTransform=affine2d([1/(repDims(1)/refDims(1)) 0 0 ; ...
                             0 1/(repDims(2)/refDims(2)) 0 ; 0 0 1]);
        else
           % Wenn das Verhaeltnis zwischen Laenge und Breite unterschiedlich ist
            scaleTransform=affine2d([0 1/(repDims(2)/refDims(1)) 0 ; ...
                             1/(repDims(1)/refDims(2)) 0 0 ; 0 0 1]);
        end
    end
```

Abbildung 30

Aufzeichnung entspricht, berechnet. Darauf findet die Transformation der Koordinaten mit den vorliegenden Transformationsdaten statt und wird angewendet (s. Abb. 30):

Für die geometrische Transformation werden die Punkte des Matching aus dem Referenzbild „matchedrefpts“ auf die Punkte des Matching aus der aktuellen Kamerabildaufnahme „matchedcampts“ abgebildet. Dies gibt einen Vektor zurück, der jedes Punktpaar, das einen Transformationsvektor bildet, entweder als Ausreißer oder als gemeinsame Menge aller Transformationsvektoren spezifiziert. Diese Transformation lässt sich mit der, in Kapitel 2. vorgestellten, integrierten Matlab Funktion „estimateGeometricTransform2D“ schätzen. Als Input kommen die Matching Punkte und die Angabe der Transformationsmethode zum Tragen. In diesem Fall werden „matchedrefpts“ und „matchedcampts“ sowie „‘projective‘“ eingesetzt, wobei letzteres den Transformationstyp einer projektiven Transformation festlegt. Der zurückgegebene Output wird als „refTransform“ und „inlierindex“ gespeichert. Bei „refTransform“ handelt es sich um die Transformationsmatrix. Im Fall des Eingangsargument „‘projective‘“ wird ein in Matlab sogenanntes „projective2d“ Objekt [37] berechnet, das aus einer 3x3 Matrix besteht und auf den Koordinatenvektor angewendet werden kann um die neuen Bildkoordinaten zu erhalten. Beispielsweise in dieser Form: $[x\ y\ 1] = [u\ v\ 1] * T$
Wobei $\mathbf{T}$ der Transformationsmatrix entspricht und $\mathbf{u}$ und $\mathbf{v}$ den Bildkoordinaten. $\mathbf{T}$ hat dabei die Form: [a b c; d e f; g h i]

Der zweite Output, „inlierindex“, ist ein binäres Mx1-Dimensionales Array, das analog zu dem Index der verwendeten Matching Punktpaare angeordnet ist. Da man theoretisch jedes Punktpaar der Matchings als Transformationsvektor nehmen könnte, aber nicht alle eine Auswirkung auf die Transformation besitzen, z.B. wenn der Vektor eine stark abweichende Resultierende im Vergleich zu allen anderen Vektoren darstellt. Jedes Element enthält entweder eine 1 (wahr), die anzeigt, dass das entsprechende Punktpaar einen Vektor innerhalb der Schätzung bildet, oder eine 0 (falsch), die anzeigt, dass das entsprechende Punktpaar einen Ausreißer bzw. unwahrscheinlicher eine Transformation von ein und demselben Punkt ist. Diese Unterscheidung ist relevant, um zufällig fehlerhafte Vektoren aufgrund von zwei matchenden Punkten, die in der Realität nichts gemein haben, sondern lediglich aufgrund der Eigenschaften gematched wurden auszuschließen. Es findet sozusagen eine Aussortierung der unbrauchbaren Vektoren statt. Wenn die aussortierten Indexe der Vektoren feststehen, speichert man diese in „inlierCameraPts“ und „inlierReferencePts“ zur Verwendung als Input für die Veranschaulichung. Da die Transformationsvektoren feststehen, können diese nun genutzt werden, um die Referenzbilddaten zu transformieren. In „frame“ werden die Bilddaten des Referenzmediums neu geladen. Als nächstes werden die Bildgrößen der Referenz, „refdims“, und des Austauschmediums, „repdims“, bestimmt. Um die Transformation auch auf ein beliebiges, zum Austausch ausgewähltes, Bild anwenden zu können, muss die Skalierung der Länge und Breite berücksichtigt werden. Die ist beispielsweise notwendig, wenn das Seitenverhältnis des Austauschmediums anders als die des Markers ist. Ansonsten würde hier eine Verzerrung der Vektoren und eine ungenaue Positionierung die Folge sein. In „scaletransform“ wird insofern ein in Matlab als „affine2d“ Objekt genanntes Objekt in der Funktion „estimateScale“ erstellt und befüllt [38]. Das Objekt ist eine 3x3

Matrix, ähnlich zu „projective2d“, nur dass die letzte Matrix Spalte standardmäßig stets die Werte 0, 0, 1 besitzt. Eine Transformationsmatrix dieser Art dient lediglich der nicht perspektivischen Transformation mit nicht veränderlichen Winkeln der Seiten, der zu transformierenden Fläche.
Für die Festlegung der Koordinaten, für die Skalierung und Verhältnisanpassung des Austauschmediums, wird mit der Funktion „imref2d“ ein Objekt erzeugt. Somit wird die Pixelausdehnung des Austauschbildes auf die Bildfläche des Referenzbildes normiert und in Relation zu einem globalen Koordinatensystem gesetzt. Das lokale Koordinatensystem des Referenzbildes wird sozusagen angeglichen. Mit „imwarp()“ wird die Transformation, unter Angabe des Koordinatensystems des Referenzbildes angewendet und ein neues auf die Größe des Marker Referenzbild (links in Abb. 31) skaliertes Bild gespeichert (rechts in Abb. 31).

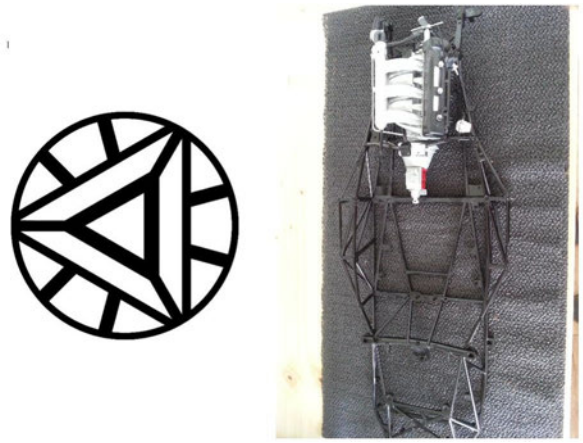


Abbildung 31

Damit gilt die Transformationsmatrix, die lediglich auf der Ansicht des Markers in der Aufnahme basiert auch für das Austauschbild und kann dementsprechend analog zur Transformation vom Marker Referenzbild zur realen Aufnahme des Markers angewendet werden. Zunächst wird „outputview“, mit der Koordinatennormierung in Bezug auf die Koordinaten der aktuellen Aufnahmen und der Festlegung der globalen Koordinaten, bedingt durch die Bildgröße der Aufnahme, überschrieben. Danach findet die Transformation des skalierten Bildes, mit der zuvor berechneten 3x3 Transformationsmatrix, unter Angabe des globalen Koordinatenbezugs „outputview“, statt. Mit „imshowpair“ wird, wie in Abb. 32 zu sehen, die vorliegende Aufnahme und das Austauschbild basierend in seiner transformierten Form dargestellt.

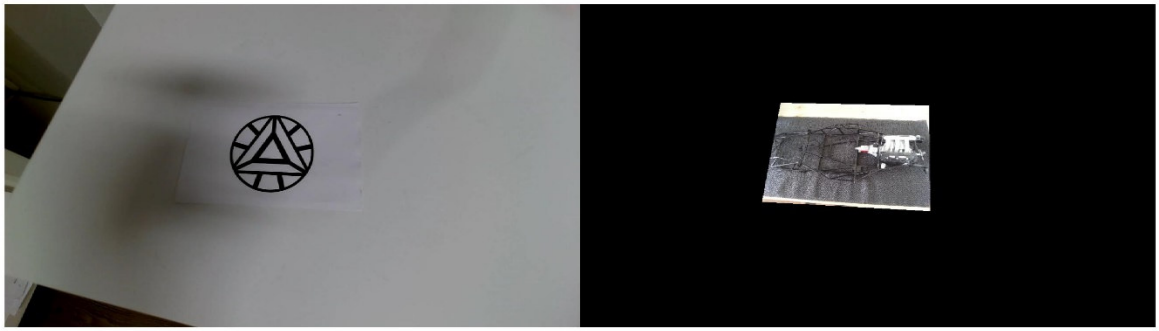


Abbildung 32

3.2.4 Darstellung/Optische Überlagerung: 2D

Jetzt, da die initiale Position der einzusetzenden, erweiternden Grafik feststeht, in nahezu den Koordinaten des erkannten Markers in einem auf das Kamerabild normierte Koordinatensystem, kann die Position direkt übernommen werden, um die beiden Bilder optisch zu überlagern.
Diese Überlagerung kann folglich ermöglicht werden:

```
%% insert transformed replace   frame into webcam frame
alphablender = vision.AlphaBlender(...
    'Operation','Binary mask','MaskSource','Input port');
mask = vidframetransformed(:,:,1) | ...
       vidframetransformed(:,:,2) | ...
       vidframetransformed(:,:,3) > 0;
outputframe = step(alphablender, currentframe, vidframetransformed,
mask);
figure;
imshow(outputframe);
```

Abbildung 33

Beginnend mit der initialisierung von „vision.AlphaBlender“ startet man den Zugriff auf ein, eigens für Matlab konzipiertes System Objekt, „System Object“, das in der Computer Vision Toolbox von Matlab inkludiert ist. - *Es handelt sich dabei um ein dynamisches Objekt in dem die Werte der Ausgangssignale sowohl von den momentanen Werten der Eingangssignale als auch von dem vergangenen Verhalten des Systems abhängen. Die vergangenen Parameter werden gespeichert, sodass diese im nächsten Berechnungsschritt zur Optimierung der Bearbeitungszeit verwendet werden können. Diese funktionale Objektstrukur ist für iterative Berechnungen optimiert, um große Datenströme in Segmenten zu verarbeiten, statt sie in einer großen Menge zu speichern und anschließend abzurufen. Die Funktion wird mit einer Iterationsanweisung aufgerufen „step()“* [39] – Der darin enthaltene Algorithmus bietet die Möglichkeit zwei Bilder zu überlagern. Der Input für die Iterationsanweisung setzt sich aus dem aktuellen Bild, „currentframe“, dem transformierten Bild, „vidframetransformed“, und einer Matrix mit binärem Inhalt in der Größe von „vidframetransformed“. Mit der „mask“ Matrix wird für jedes Pixel die Überschreibung aktiviert bzw. ignoriert je nach dem, an welcher Stelle sich ein Pixel von vidframtransformed im globalen Koordinatensystem des Bildes, das überschrieben wird. Dafür müssen die einstellbaren Parameter wie in Abb. 33 erfolgen. In „outputframe“ wird das, durch die Iterationsanweisung „step()“, berechnete Ergebnis und damit überlagerte Gesamtbild gespeichert. Somit erfolgt mit Abschluss dieser Anweisung die erste Erweiterung der Realität für die erste Aufnahme (Abb. 34).

Abbildung 34

3.2.5 Tracking: 2D

Nachdem erfolgreich eine Überlagerung für das erste Bild einer bewegten Aufnahme bzw. einer einzelnen initialen Aufnahme umgesetzt wurde. Muss sich die Überlagerung mit der Bewegung des Bildes der realen Umgebung ändern. Hierfür wird der das Tracking in einer Schleife umgesetzt, wie bereits im Konzept avisiert. Die Umsetzung in Matlab sieht dementsprechend, ergänzend zu vorangegangener Erkennung, derart aus:

```
%% initialize point tracker S
close all;
pointtracker = vision.PointTracker('MaxBidirectionalError',3,...
                                   'MaxIterations',50);
initialize(pointtracker, inlierCameraPts.Location, currentframe);
% Disp points being used for tracking
trackingmarkers = insertMarker(currentframe, inlierCameraPts.Loca-
tion,...
                  'Size',7,'Color','yellow');
figure;
imshow(trackingmarkers);
%% track Pkt auf nächstes Bild
    counter = 0;
    while counter <= 100
       %% naechstes Kamerabild
           currentframe = snapshot(cam);
          % neue, gültige tracking pkt. finden
          [trackedpoints, isValid] = step(pointtracker, currentframe);
           % nur die Pkt übernehmen die exakt getracked wurden
           newvalidlocs = trackedpoints(isValid,:);
           oldvalidlocs = inlierCameraPts.Location(isValid,:);
           trackingmarkers = ...
                  insertMarker(currentframe, newvalidlocs,...
                        'Size',7,'Color','yellow');
            imshow(trackingmarkers);
            % Reset pointtracker für nächstes Bild
            setPoints(pointtracker, newvalidlocs);
            counter = counter + 1;
    end
```

Abbildung 35

Dabei wird zuerst das System Objekt „vision.Pointtracker“ initialisiert und als „pointtracker“ gespeichert“. Daraufhin werden in „trackingmarkers“ die initialen, noch nicht bewegten, Punkte gespeichert und als die zu verfolgenden Punkte festgelegt und anschließend noch mit „imshow()“ auf der momentanen Aufnahme geplottet. Es wird eine Schleife gestartet, die zum Zweck der zeitlichen Begrenzung, der Dauer des Trackings zur Veranschaulichung, als bedingungsgesteuerte Schleife mit maximal 100 Iterationen ausgelegt ist. Zum Beispiel ist es sinnvoll sich diese Auslegung vorzubehalten, um die Erweiterung der Bedingungssteuerung mit Abbruchszenarien wahrnehmen zu können oder das Tracking nach erfolgreicher Erkennung von ausreichend vielen Punkten zu automatisieren. Innerhalb der Iteration wird zunächst das nächste Bild aufgenommen „snapshot(cam)“ und die Variable für die Speicherung der aktuellen Aufnahme überschrieben. Mit „step()“ wird die Iterationsanweisung an die Funktion innerhalb des „vision.PointTracker“ System Objekts gesendet und die Trackingpunkte „trackedpoints“, und damit die neuen Koordinaten ausgegeben. Mit dem Output „isValid“ wird beschrieben, welche getrackten Punkte als zuverlässig getrackte Punkte gelten. Damit kann in „newvalidlocs“ und „oldvalidlocs“ genau der Teil des Arrays der getrackten Punkte übergeben werden, der valide ist und das Array somit um die Fehlerhaften reduziert werden. Danach werden die Punkte analog zu dem Teil nach der Initialisierung und vor der Schleife dargestellt, bevor sie mit „setPoints()“ als neue Trackingpunkte für das System Objekt definiert werden, um mit der nächsten Iteration fortzufahren.

3.2.6 Berechnung der neuen Positionierung und Darstellung: 2D

Da das Tracking realisiert wurde kann man darauf aufbauend die Schleife erweitern, um die neue Positionierung aus dem Tracking abzuleiten und daraufhin die Darstellung bzw. Überlagerung erneut analog zur initialen Position zu vollziehen. Der große Unterschied liegt hierbei darin, dass sich lediglich die Transformationsmatrix, die eingangs bereits initial ermittelt wurde, ändert. Dabei wird die zuvor entwickelte Trackingschleife übernommen und nach dem tracken der Punkte der Vorgang der neuen Positionsberechnung vollzogen, indem nach der Auswahl der validen Trackingpunkte, die bereits zuvor verwendete Funktion „estimateGeometricTransform2D“ angewendet wird. Die neuen und alten, vorangegangen, Trackingpunkte (Im Fall des ersten Durchlaufs sind die vorangegangen Punkte die initialen Trackingpunkte vor Beginn der Schleife) stellen demnach die Eingabe dar. Ebenso identisch zu vorheriger Umsetzung des reinen Trackings werden die neuen Positionen als neue Trackingpunkte festgelegt „setPoints()“. Was hier zusätzlich nach dem Tracking erfolgt ist die Überschreibung von „trackingtransform“ und Multiplikation der Matrix mit der Transformationsmatrix der initialen Position um die globale absolute Positionierung erneut festlegen zu können, da nach dem tracken von einer alten zur nächsten Position lediglich die relative Matrix vorliegt.

Die erweiterte Lösung, der Trackingschleife, in Matlab sieht demnach folglich aus:

```
%% track Pkt auf nächstes Bild
    counter = 0;
    while counter <= 100
        %% naechstes Kamerabild
            currentframe = snapshot(cam);
            % neue, gültige tracking pkt. finden
            [trackedpoints, isValid] = step(pointtracker, currentframe);
            % nur die Pkt übernehmen die exakt getracked wurden
            newvalidlocs = trackedpoints(isValid,:);
            oldvalidlocs = inlierCameraPts.Location(isValid,:);

        %% Ermittlung d. geometr. Transformation zw. Zwei Bildern
            [trackingtransform, inldx_oldvnew]=...
                        estimateGeometricTransform2D(...
                  oldvalidlocs, newvalidlocs,'projective');% 'similarity');
            % reset Point tracker for next frame tracking
            setPoints(pointtracker, newvalidlocs);

       %% Anwendung d. geomet. transf. von 'refTransform' auf 'currentframe'
            trackingtransform.T = refTransform.T * trackingtransform.T;
            % erneute redimensionierung der ersetzenden Bilddaten
            %repframe = step(vid);
            outputview = imref2d(size(refimg));
            vidframescaled = imwarp(frame, scaletransform, ...
                                    'OutputView',outputview);
        % erneute Transformation der ersetzenden redimensionierten Bilddaten
            outputview = imref2d(size(currentframe));
                  vidframetransformed = ...
                  imwarp(vidframescaled,trackingtransform, ...
                  'OutputView',outputview);
        % insert transformed replace vid frame into webcam frame
            mask = vidframetransformed(:,:,1) | ...
                   vidframetransformed(:,:,2) | ...
                   vidframetransformed(:,:,3) > 0;

       %% Darstellung der naechsten Aufnahme
            outputframe = ...
                step(alphablender, currentframe, vidframetransformed, mask);
            figure(1);
            imshow(outputframe);
            counter = counter + 1;
    end
```

Abbildung 36

Wie schon in 3.2.3 und 3.2.4 umgesetzt, wird die Positionierung und anschließende Überlagerung übernommen. Die Bewegung der virtuellen Darstellung entsteht durch das Überschreiben der Werte mit jeder Iteration und ist lediglich durch den neuen Wert von „trackingtransform“ veränderlich. Das vollständige Skript zur Umsetzung einer einfachen Augmented-Reality Umgebung für die optische Überlagerung von 2D-Objekten lässt sich im **Anhang B** als gesamte Lösung unter dem Namen **AR_processing_Mk1_2D.m** betrachten. Die initiale Erkennung ist dabei in einer Funktion ausgelagert, um diese allgemeingültig festzuhalten.

3.3 Erstellung einer einfachen Augmented-Reality Umgebung für die optische Überlagerung von 3D-Objekten/CAD Daten

Um dem festgelegten Konzept gerecht zu werden wird in diesem Abschnitt die Erstellung einer Erweiterung der zuvor entwickelten Lösung thematisiert und entwickelt. Außerdem werden die direkt übernehmbaren Bestandteile vorgestellt und gegeben falls deren Anpassungen. Darauffolgend werden die einzelnen benötigten Schritte untersucht und die daraus hervorgehende entwickelte Lösung vorgestellt. Am Ende findet eine Feststellung des erreichten Standes statt und die notwendigen vorzunehmenden Optimierungen, die während der Entwicklung auffällig werden.

3.3.1 Übernahme von entwickelten Ansätzen des 2D Beispiels

Bevor jeder Schritt im Einzelnen für das konkrete Konzept behandelt wird, kann eine Übersicht der Bestandteile, aus der 2D AR Entwicklung, erstellt werden, die direkt adaptiert werden können:

Bei der Initialisierung steht fest, dass der **Zugriff auf die Kamera Hardware („webcam“)** übernommen werden kann, weil dessen Funktionsweise die einfachste Methode für das Bilddaten Streaming bietet. Die grundlegende **Erkennungsroutine** deckt bereits die Anforderungen für die Erkennung des Markers und bedarf keiner Anpassungen, da die Erkennung von Bildmerkmalen exakt identisch als Grundlage für das Auslösen der weiteren Prozeduren fungieren soll. Das System Objekt des **Trackings „vision.PointTracker“** kann ebenso initialisiert und eingesetzt werden. Da die Verfolgung auf den gleichen Merkmalen wie zuvor auch basiert und somit unverändert bleibt.
Übrig bleibt die initiale Position, der CAD/Austauschmedium Datenzugriff, die Berechnung der neuen Positionierung und die Darstellung als nicht adaptierbare Komponenten aus der ersten Entwicklung. Die Herausforderungen und neuen Alternativen für die genannten Bestandteile, aufgrund des Aspekts der Erweiterung der realen Aufnahme, um ein dreidimensionales virtuelles Objekt, den CAD Daten, werden in 3.3.2 vorgestellt und entwickelt.

3.3.2 Schritte bei der Bearbeitung der 3D Augmented-Reality Umgebung

Im Folgenden wird die Bearbeitung bzw. Lösung der Schritte, des Programmablaufplans des Konzepts aus 2.7 im Einzelnen behandelt, um deren jeweilige konkrete Programmierung vorzustellen. Hierbei werden die zuvor erwähnten einzelnen Ansätze und ersten Schritte eingesetzt und unter anderem auch restrukturiert, um den vollständigen Programmablauf als Matlab Skript zu erstellen. An einigen Stellen wird dies notwendig, um einen logischen Kontext zur Übergabe an den nächsten Schritt zu gewährleisten:
Wie schon erwähnt können viele Teile des Skripts, der 2D Lösung, adaptiert werden. Mit der Lösung für die Herausforderung der Initialisierung, mit der initialen Position und dem Auslesen der Daten mit anschließender Visualisierung, wurden damit die wesentlichen Anpassungen in

diesem Teil des Ablaufs erläutert. Jedoch soll an dieser Stelle zur Vervollständigung der Abläufe vor Beginn der Trackingschleife die Lösung der Erkennung aufgegriffen werden.

Herausforderungen von 2D zu 3D:

Der grundlegende Ablauf der Erkennung und des Trackings sind gemäß Programmablaufplan des Konzepts umzusetzen, wobei, wie bereits in 3.3.1 beschrieben, der allgemeine Ablauf durch die 2D Lösung bereits begründet ist. Dennoch gibt es Herausforderungen bei der Positionierung des Objekts, die durch die CAD Daten Darstellung entstehen.

Die Initialisierung ist hier ein Teil, der von der 2D Variante abweicht. Man kann zunächst die gleiche Initialisierung für die Hardware, der Kamera, verwenden, jedoch muss für das Laden bzw. Speichern des Austauschmediums ein größerer Aufwand vollzogen werden. An dieser Stelle benötigt man nicht nur eine Matrix mit den Bilddaten der Pixel, sondern eine mehrdimensionale Speicherung, welche die Daten einer beliebigen dreidimensionalen geometrischen Anordnung in einem gängigen CAD Format ermöglicht. Hierfür kommt die bereist kurz vorgestellte Funktion „STL File Reader“ zum Einsatz (weitere Details hierzu befinden sich in den Kommentarzeilen des Skripts im Anhang C und in [22]).

Der Input bildet der String eines Dateipfades auf einem lokalen Speicherort, auf den die Matlab Instanz Zugriff besitzt. In diesem Fall auf dem lokalen Laufwerk des Desktop Gerätes. Für Testzwecke zur Entwicklung wird vorerst ein vordefinierter Standard Pfad gewählt, ohne manuelle Auswahl der Datei. Der ausgelesene STL Datensatz wird als zusammengesetzte Datentypdeklaration, in einer gruppierten Liste von Variablen unter einem Namen in einem Speicherblock definiert „stl_model“. In Matlab wird dies ein „structure Array“ oder auch „struct“ genannt. Dabei besteht dieses „struct“ konkret hier aus zwei Unterstrukturen „stl_model.faces“ und „stl_model.vertices“ wie u.s. zu sehen.

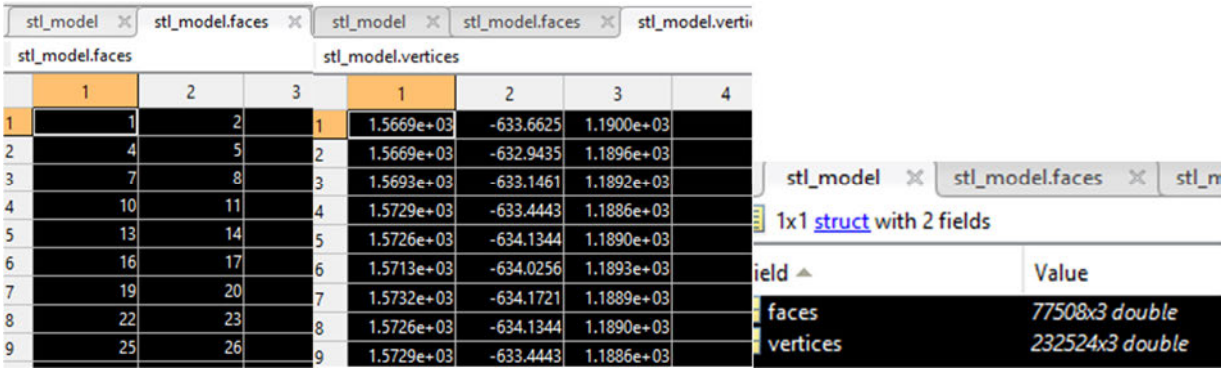


Abbildung 37

Mithilfe der Matlab internen Funktion „patch()“ kann „stl_model“ ausgelesen werden, indem es die Matrizen innerhalb der Struktur in der in Abb. 37 zu sehenden Form interpretiert. Diese Funktion erzeugt aus den Daten patch Flächen, die gleichzeitig geplottet werden können. Hierfür muss als Input jedoch zusätzlich eine Positionierungsangabe erfolgen, sowie mehrere Parameter Einstellungen. Aus diesem Grund entsteht ein weiteres Erfordernis, ein dreidimensionales Koordinatensystem innerhalb der Bildfläche des aktuellen Plots. Als erstes muss also ein Bildkoordinatensystem und danach ein zweites Koordinatensystem für die patch Flächen bzw. das CAD

Objekt definiert werden. Dafür verwendet man den Aufruf der Funktion „axes()“. Hier werden die erzeugten Koordinatensysteme in „ax1“ bzw. „ax2“ gespeichert. In dem nachstehenden Auszug des vollständigen Skripts sind die notwendigen Schritte vor Einleitung der Schleife bzw. die vollständige Initialisierung während der ersten Aufnahme abgebildet (Abb. 38):

```
%% Figure Objekt 'ar_plot' generieren und Festlegung der Eigenschaften
ar_plot = figure('Name','AR-CAD Visualizer',...
                 'NumberTitle','off',       ...
                 'MenuBar','none',          ...
                 'ToolBar','none');
ax1 = axes();
%% CAD Daten importieren: Ausgewaehlten STL Datensatz laden
stl_model = ...
    stlread('C:\Users\HAW\Bachelorarbeit_local\resources\Moeh-
ler_KK3_Hausarbeit_Dachrahmen.stl');
%skaliert die Achsen proportional zur Aufnahm0e
axis image;
hold on;
ax2 = axes('Visible', 'off');
cadpatch = patch(stl_model, ...
                'FaceColor',[0.7 0.8 1.0], ...
                'EdgeColor','none',        ...
                'FaceLighting','gouraud',  ...
                'AmbientStrength', 0.15,   ...
                'Parent', ax2);
```

Abbildung 38

Eine weitere Besonderheit stellt die Positionierung des geplotteten CAD Objekts dar. Da die Funktion das Modell standardmäßig im Ursprung des angegeben Koordinatensystems platziert, welches wiederum standardmäßig in der Mitte bzw. im Ursprung des Bildkoordinatensystems platziert wird, muss eine Berechnung der Marker Mitte erfolgen, um die genauen Ausgangskoordinaten für die initiale Positionierung zu erhalten und das Koordinatensystem darauf auszurichten. Um die Mitte zu bestimmen wird wie folgt im Skript vorgegangen:

```
%% Dimensionen der Kameraufloesung speichern
% (fuer darauffolgende dimensionierungen)
x_cam_res = str2double(extractBefore(cam.Resolution,'x'));
y_cam_res = str2double(extractAfter(cam.Resolution,'x'));

initialize( ...
    pointtracker, matchedcampts.Location, currentframe);
    [trackedpoints, isValid] = ...
    step(pointtracker, currentframe);
    currentvalidlocs = trackedpoints(isValid,:);
    mean_middlepoint = mean(currentvalidlocs,1);
% Mittelpunkt speichern
middlepoint = ...
    [mean_middlepoint(1,1)/x_cam_res ...
     mean_middlepoint(1,2)/y_cam_res];
ax2.Position = [middlepoint(1,1)-0.34 middlepoint(1,2)-0.17 0.77 0.34];
```

Zuerst wird die Auflösung in x- und y-Richtung ermittelt und gespeichert. Daraufhin wird über alle getrackten Punkte nach der initialen Erkennung bzw. den aus der Erkennung abgeleitet Trackingpunkten, das arithmetische Mittel der Bildkoordinaten in Pixeleinheiten berechnet „mean_middelpoint“. Dieser, sich daraus ergebende Punkt, bildet den geometrischen Schwerpunkt des Markers ab und liegt in jedem Fall, sofern es sich um auf dem Marker befindliche Koordinaten handelt, in jedem Fall innerhalb der Markerfläche (s. Abb. 39).

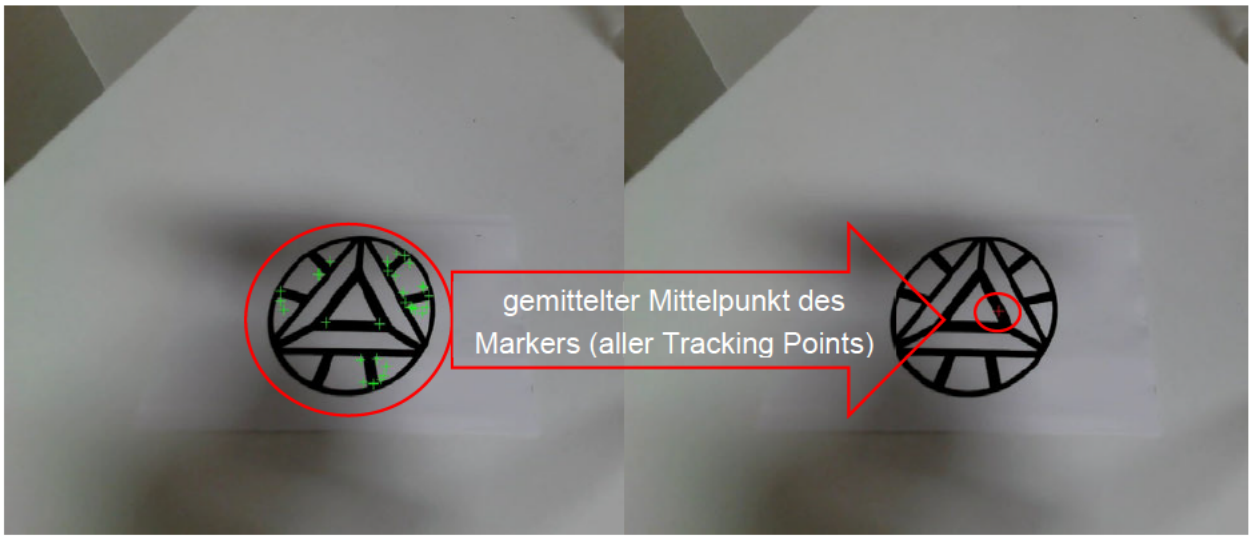


Abbildung 39

Mit der Anzahl der Pixel in beide Ausdehnungen des Bildes lässt sich dies in die Skalierung des Bildkoordinatensystems „ax1“ umrechnen und der absolute Mittelpunkt des Markers in Bildkoordinaten speichern. Das CAD Objekt Koordinatensystem „ax2“ stellt dabei ein lokales Koordinatensystem innerhalb der Bildfläche und damit innerhalb des globalen Koordinatensystems „ax1“ dar. Damit das lokale Koordinatensystem genau an der Stelle der Marker Mitte auf dem Bild zu sehen sein wird muss die Position/Ursprung mit den Mittelpunkt Koordinaten angegeben werden „middlepoint“. Wie in Abb. 40 zu sehen ist die Positionierung des CAD Objekts bzw. die Überlagerung mit einer Aufnahme der Realität durch das Überlagern des Bildkoordinatensystems (ganzes Bild) und dem lokalen Objekt Koordinatensystem realisiert.

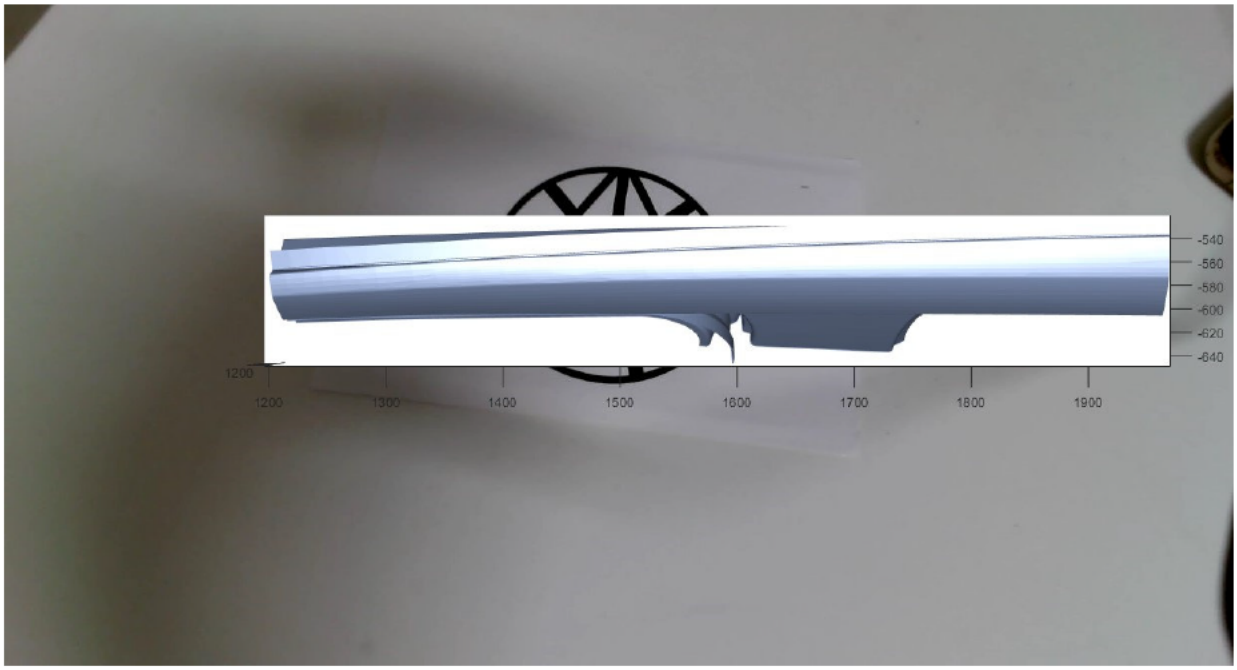


Abbildung 40

Zur vollständigen optischen Einfassung des Objekts in die Realitätsaufnahme, werden nur die CAD Flächen gerendert und mit der Aufnahme überlagert (s. Abb 41).

Abbildung 41

Bei der Vorgehensweise wie man die Berechnung der neuen Positionierung vollzieht, wird genau wie im 2D Beispiel die initiale Positionierung modifiziert und damit die ursprüngliche Position überschrieben. Der Vorgang der Positionierung, des CAD Objekt Koordinatensystems „ax2.Position“, wird in Abhängigkeit der Transformationsvektoren, die aus dem Tracking hervorgehen iteriert. Somit soll sich der Ursprung parallel mit den Trackingpunkten bewegen.

Implementieren der Markermittelpunkt Bestimmung in die Erkennung:

Zu Beginn wird das Kameraobjekt wie zuvor mit der Funktion „webcam“ initialisiert. Darauffolgend werde alle Methoden, die in der Lösung der Herausforderungen von 2D zu 3D gelöst wurden, übernommen.

```
%%
cam = webcam;
%% CAD Daten importieren: Ausgewaehlten STL Datensatz laden
stl_model = ...
    stlread('C:\Users\HAW\Bachelorarbeit_local\resources\Moehler_KK3_Hausar-
beit_Dachrahmen.stl');
 %% initialize point tracker
pointtracker = vision.PointTracker('MaxBidirectionalError',3,...
                                   'MaxIterations',50);
%% Dimensionen der Kameraufloesung speichern
% (fuer darauffolgende dimensionierungen)
x_cam_res = str2double(extractBefore(cam.Resolution,'x'));
y_cam_res = str2double(extractAfter(cam.Resolution,'x'));
%%  matching der features d. referenzbilddaten mit den kamerabilddaten
currentvalidlocs = 0;
firstrun = 1;
%% Figure Objekt 'ar_plot' generieren und Festlegung der Eigenschaften
ar_plot = figure('Name','AR-CAD Visualizer',...
                 'NumberTitle','off',       ...
                 'MenuBar','none',          ...
                 'ToolBar','none');
ax1 = axes();
%axis image:
```

```
%Def.: "Sets DataAspectRatio to [1 1 1] and sets the associated mode
%       property to manual"
%skaliert die Achsen proportional zur Aufnahm0e
axis image;
hold on;
ax2 = axes('Visible', 'off');
%% polygon Flaechen des STL Datensatzes in Koord. des Figure Objekts plot-
ten;
cadpatch = patch(stl_model, ...
                 'FaceColor',[0.7 0.8 1.0], ...
                 'EdgeColor','none',        ...
                 'FaceLighting','gouraud',  ...
                 'AmbientStrength', 0.15,   ...
                 'Parent', ax2);
%axis vis3d matlab def.: stellt den Modus des plot-Raum Seitenverhaeltnis
%und des Daten Seitenverhaeltnis auf manuell -> inidviduell definierbare
%Raum Dimensionierung ermoeglicht 3D Effekte
axis vis3d;
axis image;
camlight('headlight');
camproj('perspective');
```

Mit dieser modifizierten Initialisierung der Randbedingungen und virtuellen Objektdaten, ist an dieser Stelle die Grundvoraussetzung für die Weitergabe an eine angepasste Erkennung, und damit die Initialisierung des SURF Detektors geschaffen. Es wird die gleiche Art der Erkennung, wie bereits in 3.2 thematisiert, angestrebt. Aus diesem Grund wird dieser Teil in seiner grundlegenden Vorgehensweise übernommen, jedoch in eine Funktion ausgelagert, um die Verwendung der Erkennung als alleinstehenden Bestandteil zu generalisieren. Mit dieser Auslagerung kann die Erkennung modularer eingesetzt werden und ist auf weitere Entwicklungen adaptierbar. Der Input besteht aus der Angabe des „pointtracker“ System Objekts, dem Kameraobjekt „cam“ und dem Bildkoordinatensystem „ax1“.

```
function [mean_middlepoint, currentvalidlocs, ...
              currentframe, refimg, validpts, ...
              camFeatures, validrefpts,reffeatures] = ...
              initital_detection(pointtracker, cam, ax1)
      % Initi. Referenzbild 'refimg' und Bildmerkmale erfassen:
      refimg = imread('firstimg.jpg');
      % erkennen + extrahieren der 'SURF Features'
      refimggray = rgb2gray(refimg);
      refpts = detectSURFFeatures(refimggray);
      refpts = refpts.selectStrongest(75);
      [reffeatures, validrefpts] = ...
                extractFeatures(refimggray, refpts);
      % Darstellung der 'SURF Features'
      % init. kamera 'cam' und Bildmerkmale erfassen:
      % verbindung zur Kamera aufbauen
      % erkennen + extrahieren der 'SURF Features' in aktueller Aufnahme
      for i=1:10
            currentframe = snapshot(cam,'immediate');
            currentframegray = rgb2gray(currentframe);
            campts = detectSURFFeatures(currentframegray);
            campts = campts.selectStrongest(75);
            % darstellung der akt. Aufnahme
            imshow(currentframe,'Parent', ax1,'Border','tight');
            [camFeatures, validpts] = ...
                  extractFeatures(currentframegray, campts);
```

```
            % Matching der Features zwischen Aufnahme und Referenzbild
            idxpairs = ...
            matchFeatures(camFeatures, reffeatures,'MaxRatio',0.9);%,...
            %'Method','Approximate','MatchThreshold',15);
            % speichern der jwl. matched surf pkt.
            matchedcampts = validpts(idxpairs(:,1));
            plot(validpts.selectStrongest(75));
        end
        release(pointtracker);
        initialize(pointtracker, matchedcampts.Location, currentframe);
        [trackedpoints, isValid] = ...
                    step(pointtracker, currentframe);
        %Übernahme der validen Trackingpunkte
        currentvalidlocs = trackedpoints(isValid,:);
        %Berechnung des arithm. Mittels der Koordinaten
        mean_middlepoint = mean(currentvalidlocs,1);
        imshow(currentframe,'Parent', ax1,'Border','tight');
end
```

Wie bereits entwickelt wird das Referenzbild und das aktuelle Kamerabild mit der „detectSURFFeatures“ Funktion untersucht. Innerhalb der Funktion ist das Referenzbild für den Marker statisch definiert. Eine Erweiterung zu einer variablen Auswahl eines beliebigen Markers ist jedoch bei dieser Funktionsmethode einfach zu ergänzen, indem man diesen ebenfalls als input definieren würde und das Marker Referenzbild bei der Initialisierung festlegt. Mit in die Erkennungsfunktion „initial_detection“ ist außerdem die Initialisierung und Anwendung der ersten Iteration des „pointtracker“ System Objekts und die vorgestellte Bestimmung des gemittelten geometrischen Schwerpunkts „mean_middlepoint“ integriert. Mit den zuvor festgelegten Werten als Input erhält man den „mean_middlepoint“ Markermittelpunkt und die zur Berechnung verwendeten erkannten Features und zugehörige Koordinaten. Anschließend kann die Funktion wie folgt im gesamten Kontext eingesetzt werden:

```
%% inititalisierung des SURF Detektors und Erkennung des Markermittelpunkts
[mean_middlepoint, currentvalidlocs, ...
 currentframe, refimg, validpts, ...
 camFeatures, validrefpts,reffeatures] = ...
 initial_detection (pointtracker, cam, ax1);
% Mittelpunkt speichern
middlepoint = ...
    [mean_middlepoint(1,1)/x_cam_res ...
     mean_middlepoint(1,2)/y_cam_res];
% initiale Positionierung des ax2 Koordinatensystem Ursprungs
ax2.Position = [middlepoint(1,1)-0.34 middlepoint(1,2)-0.17 0.77 0.34];
```

Mit den Werten in „mean_middlepoint“ lässt sich dieser, mit den Werten zur Bildauflösung, in Bildpixelkoordinaten umrechnen, sodass die Koordinaten passend für die Positionierung von „ax2“ in den Koordinaten von „ax1“ positioniert werden können. Mit diesem Schritt „ax2.position“ ist die initiale Positionierung vollzogen und somit alle Schritte vor dem Einleiten der Trackingschleife absolviert. Das Tracking wird von der 2D Lösung übernommen und daraufhin angepasst, sodass eine Transformation als Berechnung der neuen Positionierung des CAD Objekts erfolgt. Es fallen jedoch Besonderheiten auf, die einer Anpassung zur neuen Positionierung bedürfen:

```
%% merkmalspunkte auf naechstes Bild tracken
      % Iteration fuer eine festgelegte Anzahl an Durchlaeufen
      % Das Tracking der Punkte erfolgt nach jeder erneuten Aufnahme
      % Der komplette Durchlauf erzeugt ein Bewegtes Bild und ermittelt
      % durchgehend die Transformation der Trackingpunkte
   % default Werte
isValid = 0;
counter = 0;
while counter <= 500
      % speichern des vorangegangenen Kamerabildes
      prevcurrentframe = currentframe;
      % vorangegangenen Durchschnittsmittelpunkt speichern
      prevmiddlepos = mean_middlepoint;
      prevmiddlepos = [
                       prevmiddlepos(1,1)/x_cam_res-0.34 ...
                       prevmiddlepos(1,2)/y_cam_res-0.17
                      ];
    % naechstes Kamerabild
    currentframe = snapshot(cam);
    % neue, gültige tracking pkt. finden
    [trackedpoints, isValid] = step(pointtracker, currentframe);
    % vorangegange tracking pkt. speichern
    prevvalidlocs = currentvalidlocs(isValid,:);
    % nur die neuen Trackingpkt. übernehmen die exakt getracked wurden
    currentvalidlocs = trackedpoints(isValid,:);
    % Ermittlung der Transformationsmatrix zwischen Zwei Trackingkoordinaten
    [trackingtransform, inlierindx_oldtonew]=...
                estimateGeometricTransform2D(...
                prevvalidlocs, currentvalidlocs,'projective');
    %disp(trackingtransform.T);
    oldinlierloc = prevvalidlocs(inlierindx_oldtonew,:);
    newinlierloc = currentvalidlocs(inlierindx_oldtonew,:);
    %% Transformation der Ansicht bestimmen
      % Translation:
      % Bewegungsmatrix (m x n: m = Anzahl der getrackten Pkt; n = 2 (x,y) )
      trackedpoints_translatorymotion = newinlierloc - oldinlierloc;
      % arithmetisches Mittel entlang der ersten Matrix Dimension
      % m x n Matrix zu einem 1 x n Vektor
      % ->durschnittliche translatorische Bewegung der Pkt.(Bildkoordinaten)
        trackedpoints_translatorymotion = ...
            double(mean(trackedpoints_translatorymotion,1));
      % Anpassung der Skalierung
    % (basierend auf der Kameraauflösung Koordinaten des Bildes)
        trackedpoints_translatorymotion(1,1) = ...
            trackedpoints_translatorymotion(1,1) / x_cam_res;
        trackedpoints_translatorymotion(1,2) = ...
            trackedpoints_translatorymotion(1,2) / y_cam_res;
%% Transformation der Ansicht
    % Anwendung des Translationsvektors
    % (Koordinatensystem des CAD Modells neu ausrichten)
    if prevvalidlocs ~= 0
        newviewpos = prevmiddlepos + ... %campos + ...
                     [
                     trackedpoints_translatorymotion(:,1) ...
                     (-1*trackedpoints_translatorymotion(:,2))...
                     ];
        figure(ar_plot);
        ax2.Position = [
                        newviewpos ...
                        0.75 ...
                        0.75 ...
                       ];
       if patchdelete == 1
```

```
        cadpatch = patch(stl_model, ...
                         'FaceColor',[0.7 0.8 1.0], ...
                         'EdgeColor','none',        ...
                         'FaceLighting','gouraud',  ...
                         'AmbientStrength', 0.15,   ...
                         'Parent', ax2);
        end
    end
    %% Festlegung der darauffolgenden Trackingpkt.
    % (ueberschreiben des Trackers)
    setPoints(pointtracker, currentvalidlocs);
    %%Visualis. der Trackingpunkte als zusammengefassten Punkt aktualisieren
    figure(ar_plot);
    %middelpoint_marker: Wiederspiegelung des Mittelwerts der Trackingpkt.
    middlepoint_marker = insertMarker( ...
        currentframe, newviewpos, 'Size',7,'Color','red');
    imshow(middlepoint_marker,'Parent',ax1,'Border','tight');
    % iteration (Kameraufnahme-Schleife um einen erhoehen)
    counter = counter + 1;
    isValid = 0;
end
```

Wie schon aus den aufgestellten Anforderungen bekannt, muss zur Translation auch eine Rotation des Objekts bzw. der Ansicht erfolgen, um eine vollkommen räumliche Perspektive zu erzeugen. Dafür muss, mit dem gewählten Ansatz, eine Möglichkeit gefunden werden dies rein optisch zu erzielen. Zunächst fokussiert man sich hier auf die Bearbeitung der translatorischen Neu Positionierung. Hierfür wird der Ansatz der Markermittelpunkt Bestimmung weiterentwickelt, um die absolute Position des Objekts, für das Ziel, dass das Objekt stets auf dem Marker liegen soll, zu bestimmen. Die translatorische Bewegung ist eine Ableitung, der Position der Markermitte. Daher wird an dieser nicht mit der sofortigen Position der Mitte gearbeitet, sondern der Verschiebungsvektor bzw. die Bewegung ermittelt und auf die vorangegangene Position addiert. Man könnte, aus mathematischer Sicht, für jede Iteration den Markermittelpunkt berechnen über die aktuelle Position der Trackingpunkte berechnen, jedoch bietet die Bestimmung des Verschiebungsvektors mehr Möglichkeiten für das Anknüpfen an weitere Transformationsmethoden. Der Vorteile besteht darin, dass ein Verschiebungsvektor bzw. Bewegungsvektor aufgrund deiner relativen Größe zur Weiterentwicklung von beschleunigungsoptimierten Trackingansätzen genutzt werden kann. Diese Auslegung behält man sich vor und baut zunächst die translatorische Bewegung darauf auf.

Man verwendet wieder die Funktion „estimateGeometricTransform2D“. Jedoch wird die Funktion an dieser Stelle in einem anderen Kontext als bei der 2D Lösung verwendet. Hier nutzt man die Funktion vorerst nur, um die Ausreißer zu eliminieren und den Index der brauchbaren Vektoren bzw. Punkte zu bestimmen. In „trackedpoints_translatorymotion“ wird als erstes die Berechnung einer zweidimensionalen Bewegungsmatrix mit den jeweiligen Koordinatenänderungen der Trackingpunkte gespeichert. Diese Koordinatenänderungen können als Differenzen zwischen den vorherigen und den aktuellen Koordinaten gesehen werden. Dann wird über diese Matrix eine Mittelwert entlang der ersten Matrix Dimension (m: Anzahl der Punktkoordinaten Differenzen) gebildet. Mit dieser mittleren Bewegung lässt sich daraus genau ein neuer Punkt ermitteln, der dem echten Markermittelpunkt, angenähert, gleichen soll „newviewpos“. Diese

neue Position wird mit der schon bekannten Funktion „ax2.Position“ angewendet bzw. das Koordinatensystem „ax2“ neu positioniert. Mit der neuen Position des Ursprungs ist auch das Objekt an dieser Position verankert und bewegt sich mit jeder nächsten Bewegung mit.

Mit diesem Stand werden mindestens alle Anforderungen des Konzepts bedient, während sich eine CAD Objekt mit der Realität überlagern lässt und dessen Position sich mit der Ansicht in gewisser Weise synchron ändert. In der Bildsequenz in Abb. 42 wird dies einmal dargestellt:

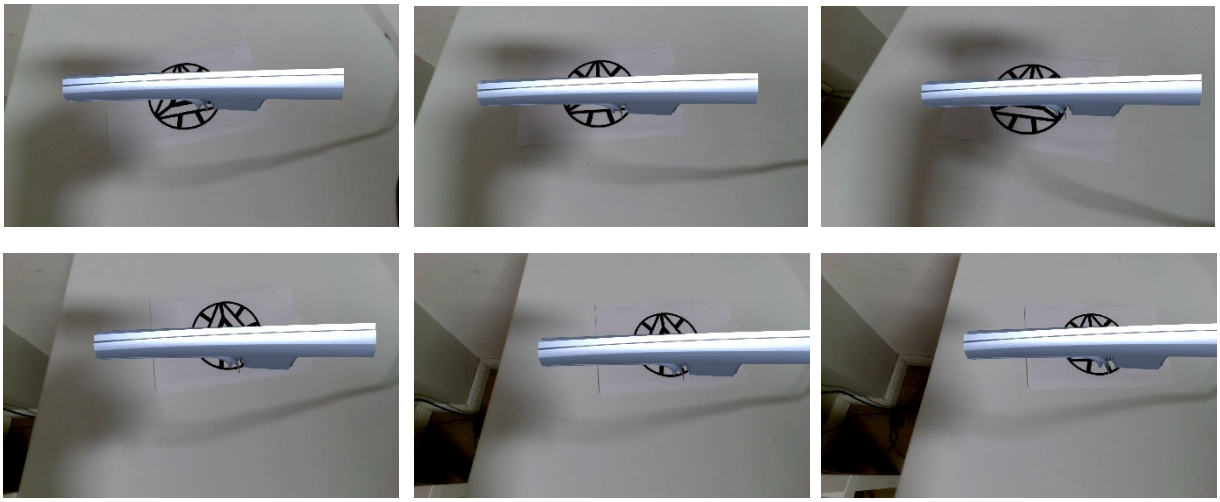


Abbildung 42

3.3.3 Zwischenergebnisse und Optimierungsschleife

Das Zwischenergebnis der erzielten Anwendung ist nach mehrfacher Testung, noch an ein einigen nicht Stabil. So kommt es z.B. vor, dass die Erkennung zwar erfolgreich stattfindet, aber das Tracking und die geometrische Transformation mindestens einen Input von 4 übereinstimmenden Features bzw. letztendlich Punkten erwartet. Ebenso problematisch ist das Verlieren der Ansicht des Markers, zum Beispiel bei einer Aufnahme, die keine Abbildung des Markers mehr beinhaltet oder dieser plötzlich durch stark veränderliche Lichtverhältnisse oder physische Manipulation der Oberfläche unlesbar wird und noch vieles mehr.
Zusätzlich besitzt dieses Ergebnis noch nicht die Fähigkeit die Rotation der Ansicht bzw. die volle perspektivische Transformation für ein noch weitaus realitätsnäheres räumliches Anwendungserlebnis. Diese Umsetzung stellt keine Optimierung dar, sondern eine mögliche Ergänzung, die das Gesamtergebnis aufwerten würde. Daher liegt der Fokus zuerst auf der Beseitigung der Laufzeitschwierigkeiten durch inkonsistente Abläufe.

Das Problem der unzureichenden Erkennung und der Abbruch während das Trackings müssen daher abgefangen werden. Es muss nach dem Scheitern des Trackings eine intermediäre, automatische Neu-Initialisierung stattfinden, um eine kontinuierliche Wiederaufnahme des Trackings zu gewährleisten. Man stelle sich hierfür folgende Szenarien vor:

1. Die Anwendung wird gestartet und erkannt und trackt erwartungsgemäß den Marker, sodass eine optische Überlagerung vollzogen werden kann. Darauf wendet der User jedoch die Sicht vom Marker ab, sodass dieser nicht mehr in der aktuellen Aufnahme zu sehen ist. Es ist demnach nur durch ein aktives Eingreifen bzw. Neustarten möglich das Tracking fortzusetzen, da die erkannten Merkmale verloren sind und nicht mehr gemapped werden können.
2. Die Anwendung wird gestartet und läuft jedoch gelegentlich in einen Rechenfehler innerhalb des System Objekts des Trackings, da dieser mind. 4 Matching Punkte erwartet. Außerdem würde die Funktion estimateGeometricTransform2D ebenfalls nicht funktionieren. Abgesehen von diesen Soll Kriterien des Inputs, der Funktionen, sind stets so viele matching Punkte wie möglich anzustreben, da sich aus ihnen ohnehin eine statistisch tragbarere Transformation ableiten lässt.

Im Skript wurde daher, als Auszug, u.s. zu sehen, zunächst der Teil der Erkennung und die Bestimmung des Markermittelpunktes noch granularer modularisiert:

```
function [mean_middlepoint, currentvalidlocs, ...
              currentframe, refimg, validpts, ...
              camFeatures, validrefpts,reffeatures] = ...
              initial_detection ( ...
                firstrun, currentvalidlocs, pointtracker, cam, ax1)
        while firstrun == 1 || size(currentvalidlocs,1) < 30
            firstrun = 0;
            clear inlierCameraPts;
            [currentframe, refimg,...
            validpts, camFeatures, ...
            validrefpts,  reffeatures, ...
            matchedcampts] = ini_imgsource(cam,ax1);
            release(pointtracker);
            if matchedcampts.Count >= 4
                initialize( ...
                    pointtracker, matchedcampts.Location, currentframe);
                [trackedpoints, isValid] = ...
                    step(pointtracker, currentframe);
                currentvalidlocs = trackedpoints(isValid,:);
                mean_middlepoint = mean(currentvalidlocs,1);
            end
        end
        imshow(currentframe,'Parent', ax1,'Border','tight');
end
%% Initialisierung der Bilddaten;Erkennung von SURF Punkten
% Extrahieren der features für die Übergabe an matchFeatures
function [currentframe,refimg,...
              validpts,camFeatures, ...
              validrefpts,reffeatures, ...
              matchedcampts] = ini_imgsource(cam,ax1)
      % Initi. Referenzbild 'refimg' und Bildmerkmale erfassen:
      refimg = imread('firstimg.jpg');
      % erkennen + extrahieren der 'SURF Features'
      refimggray = rgb2gray(refimg);
      refpts = detectSURFFeatures(refimggray);
      refpts = refpts.selectStrongest(75);
      [reffeatures, validrefpts] = ...
      extractFeatures(refimggray, refpts);
```

```
        % Darstellung der 'SURF Features'
        % init. Kamera- 'cam' und Bildmerkmale erfassen:
        % Verbindung zur Kamera aufbauen
        % erkennen + extrahieren der 'SURF Features' in aktueller Aufnahme
        matchedcampts.Count = 0;
        while matchedcampts.Count < 4
              currentframe = snapshot(cam);
              currentframegray = rgb2gray(currentframe);
              campts = detectSURFFeatures(currentframegray);
              campts = campts.selectStrongest(75);
              % darstellung der akt. Aufnahme
              imshow(currentframe,'Parent', ax1,'Border','tight');
              [camFeatures, validpts] = ...
                    extractFeatures(currentframegray, campts);
              % Matching der Features zwischen Aufnahme und Referenzbild
              idxpairs = ...
              matchFeatures(camFeatures, reffeatures,'MaxRatio',0.9);%,...
              %'Method','Approximate','MatchThreshold',15);
              % speichern der jwl. matched surf pkt.
             matchedcampts = validpts(idxpairs(:,1));
        end
end
```

In der Funktion „initial_detection“ sind weiterhin alle Bestandteile zum Erzielen der Erkennung enthalten. Jedoch wird zu Beginn bereits eine Bedingungsgesteuerte Schleife initialisiert, dessen Bedingung erfüllt ist sobald die Variable „firstrun“ den Wert 1 hat oder die Größe des Arrays der aktuell erkannten Merkmale kleiner als 30 ist. Erstere Bedingung sorgt dafür, dass bei der absolut ersten Ausführung der Funktion, die Schleife, inklusive der darin enthaltenen Erkennung und matching von Merkmalen verarbeitet wird. Im ersten Lauf der Schleife wird die Variable „firstrun“ auf 0 gesetzt. Darauf wird die Funktion „ini_imgsource“ aufgerufen. Diese Funktion stellt eine Auslagerung der reinen Erkennung, Extraktion und anschließendem Matching dar, ohne jegliche andere Berechnung. Der große Unterschied bei dieser Funktion besteht darin, dass diese stets mindestens 4 übereinstimmende Punkte aus dem Referenzbild und der Kamera Aufnahme ausgibt. Man realisiert dies durch eine stetige Wiederholung von der Erkennung bis zum Matching, solange weniger als 4 „matchedcamppts“ Übereinstimmungspunkte auf der Kameraaufnahme erkannt, extrahiert und gematched werden. Während der Erkennung werden als zusätzliche Auswahl lediglich die stärksten 75 Merkmale gesammelt. Dies basiert auf der Erkenntnis durch Testungen, in denen bei mehr als ca. 75 Merkmalen, auf einer Aufnahme, zu viele Fehlübereinstimmungen entstehen. Mit den genannten Anpassungen ist das Erlangen von weiter verwendbaren Bildmerkmalen vollkommen und funktioniert autark in einem beliebigen Kontext. Die Voraussetzung ist ein Kameraobjekt und ein Bildkoordinatensystem aufgrund der Skalierung des aufgezeichneten Bildes als Input. Es ist wichtig hierbei nicht außer Acht zu lassen, dass ein variabler Input für ein auszuwählendes Referenzbild stets als Input aufgenommen werden kann. Jedoch wird bei dieser Entwicklung das Referenzbild vordefiniert und in jedem Vergleich beibehalten. Zurückblickend auf die Stelle in der Funktion „initial_detection“ in der die Funktion aufgerufen wird, erhält man nach diesem Prozedurschritt die Merkmalspunkte. Diese können dann, nach der Initialisierung des Tracking System Objekts an die Verarbeitung der Merkmale als Trackingpunkte im fortlaufenden Prozess übergeben werden. Am Ende der Funktion findet wie schon vorher umgesetzt die Berechnung des mittleren Marker Mittelpunkts aus den genannten, von hier an definierten Punkten, statt. Mit diesem Umbau der Funktion wird bereits 2. der

Laufzeitprobleme behoben, indem es gar nicht erst zu einer voreiligen, unzureichenden Initialisierung kommt. Der Output dieser Funktion besteht nun aus dem mittleren Marker Mittelpunkt „mean_middlepoint“, den aktuellen validen Trackingpunkten „currentvalidlocs“, der aktuellen Aufnahme „currentframe“, dem Referenzbild „refimg“, den validen extrahierten Punkten der Aufnahme „validpts“, den Kamera Features/Merkmalen „camFeatures“, den validen extrahierten Punkten des Referenzbildes „validrefpts“ und den Referenzbild Features/Merkmalen „reffeatures“.

Nun, um anschließend noch das 1. der beiden Probleme zu beseitigen, wird die Funktion derart in das laufende Tracking eingebunden, dass während eines voraussichtlich fehlerhaften nachfolgenden Trackings kein Laufzeitfehler entsteht, sondern mit einer Bedingung zu jeder Iteration des Trackings potenziell abgefangen werden kann. Zu Beginn der Trackingschleife findet direkt eine Bedingungsüberprüfung statt, noch bevor die nächsten Trackingpunkte definiert werden. Wie hier zu sehen, wird dies in die bereits bestehende Routine implementiert und neu strukturiert. Dabei werden den Variablen „isValid“ und „counter“ die Standardwerte 0 zugewiesen.

```
%% merkmalspunkte auf naechstes Bild tracken
% Iteration fuer eine festgelegte Anzahl an Durchlaeufen
% Das Tracking der Punkte erfolgt nach jeder erneuten Aufnahme
% Der komplette Durchlauf erzeugt ein Bewegtes Bild und ermittelt durchge-
hend
   % die Transformation der Trackingpunkte
   % default Werte

isValid = 0;
counter = 0;

while counter <= 500
    % Initialisierung fuer die erste Erkennung
    % (Schleife zur Aufrechterhaltung bis der Marker erneut gefunden wird)
    while sum(isValid) < 4
        patchdelete = 0;
        while size(currentvalidlocs,1) < 4
            if firstrun == 1
                firstrun = 0;
            else
                patchdelete = 1;
                delete(cadpatch);
            end
            [mean_middlepoint, currentvalidlocs, currentframe] = ...
                 initial_detection( ...
                       firstrun, currentvalidlocs, pointtracker, cam, ax1);
        end
        % speichern des vorangegangenen Kamerabildes
        prevcurrentframe = currentframe;
        % vorangegangenen Durchschnittsmittelpunkt speichern
        prevmiddlepos = mean_middlepoint;
        prevmiddlepos = [
                         prevmiddlepos(1,1)/x_cam_res-0.34 ...
                         prevmiddlepos(1,2)/y_cam_res-0.17
                        ];
        % naechstes Kamerabild
```

```
        currentframe = snapshot(cam);
        % neue, gültige tracking pkt. finden
        [trackedpoints, isValid] = step(pointtracker, currentframe);
        % vorangegange tracking pkt. speichern
        prevvalidlocs = currentvalidlocs(isValid,:);
        % nur die neuen Trackingpkt. übernehmen die exakt getracked wurden
        currentvalidlocs = trackedpoints(isValid,:);
    end
    % neuen Durchschnittsmittelpunkt bestimmen
      .
      .
      .
end
```

Wie hier zu sehen, wird bei weniger als vier validen Trackingpunkte die Bedingungsgesteuerte Schleife eingeleitet. Aufgrund der Standard Einstellung beim ersten Durchlauf wird dies durchlaufen. Die Variable „patchdelete“, die als boolesche Variable für das Löschen der CAD Daten Patch Objekts aus der aktuellen Aufnahme steht, wird auf 0 gesetzt. Der Hintergrund zu dieser Variablen wird zu einem späteren Punkt erläutert. In einer zweiten „while“-Schleife wird ergänzend geprüft, ob die tatsächliche Größe des Arrays von „currentvalidlocs“ bzw. die Anzahl aller validen getrackten Punkte kleiner als vier ist. Ist dies der Fall wird die Schleife initiiert. Diese Schleife wird verwendet, um das Tracking nach einem Ausbruch aus der Schleife schneller zu ermöglichen, indem keine vergangenen Trackingpunkte, die eventuell sofort wieder überschrieben werden müssen nochmals in der Hauptprozedur gespeichert werden. Also entsteht durch die zweite Schleife eine Segmentierung zur Laufzeitverbesserung, ohne die es während der Testung zu schlechten Frameraten kam, die den kontinuierlichen Bewegungseindruck der Aufnahme verringern.
Am Ende erzielt diese Optimierung, bei der eine Re-Initialisierung während der Laufzeit ermöglicht wird, ein fortlaufendes stabileres Erlebnis in der Augmented Reality Umgebung und verbessert die Handhabbarkeit. Mit diesem Stand ist ein geeigneter Prototyp für eine Testumgebung geschaffen, der, unter Vorbehalt der zu erweiternden geometrischen Transformation für die Ansichtsrotation, im Release fähigen Detail weiterentwickelt werden kann. Der hier vorgestellte Prototyp ist im Anhang C mit dem Dateinamen **AR_processing_Mk2_3D.m** hinterlegt.

3.4 Implementierung der Endversion auf einem handelsüblichen Smartphone

Mit dem Prototyp, der auf der Desktop Version von Matlab funktioniert, gilt es diesen in eine, auf einem handelsüblichen Smartphone laufenden, Anwendung zu implementieren. Daher wird an dieser Stelle mit den wichtigsten Vorbereitungen begonnen und zunächst versucht alle im prototypischen Matlab Skript enthaltenen Funktionen in C zu generieren. Aufgrund einer mehrfachen Testung und Verifizierung mit jeweils folgender Anpassung, wird der Fokus, im Rahmen dieser Arbeit nur darauf gelegt das konzeptionelle Vorgehen kurz festzuhalten und an die Leser weiterzugeben.
An sich besteht grundsätzlich die Möglichkeit einen Matlab Code auf der, für Matlab User erhältlichen, Matlab Mobile App auszuführen. Dennoch muss der Code iterativ angepasst werden bis er vollständig adaptierbar auf ein iOS oder Android System in der Matlab Mobile App ist. Es

fällt jedoch bei der Testung auf, dass die Matlab Mobile App nicht alle Funktion und direktiven der Desktop Version vollumfänglich unterstützt. Damit ist keine hinreichende Kompatibilität gewährleistet, durch die eine Code Generierung in eine andere Programmiersprache zwingend notwendig wird. Es bietet sich zum Beispiel C an, da man hiermit eine Implementierung mithilfe der XCode IDE, für iOS Betriebssysteme, oder der Android SDK, für Android Betriebssysteme, als App erzielen kann. Die Code Generierung in C lässt sich durch ein verfügbares Add-On in Matlab realisieren, den „Matlab Coder“ [40]. Mit diesem Tool können Matlab Funktion bzw. daraus bestehende Skripte als C Code generiert werden. Bei der Anwendung des Tools gibt es bereits vereinfachte Annahmen und Definitionen von Funktionen, sodass der Umgang mit dem Tool leicht umgänglich ist. Es gibt jedoch auch einige nicht unterstützte Methoden und Funktionen bei der Generierung (s. Abb. 43). So ist zum Beispiel eine Generierung, einer äquivalenten Funktion in C, der Funktion „webcam“, der Bilddaten Erfassung durch eine optische Hardware des Systems nicht möglich, da diese nicht im Generierungstool berücksichtigt wird. Dies gilt auch für weitere **Funktionen, die ohne neu Codierung nicht generierbar sind**.

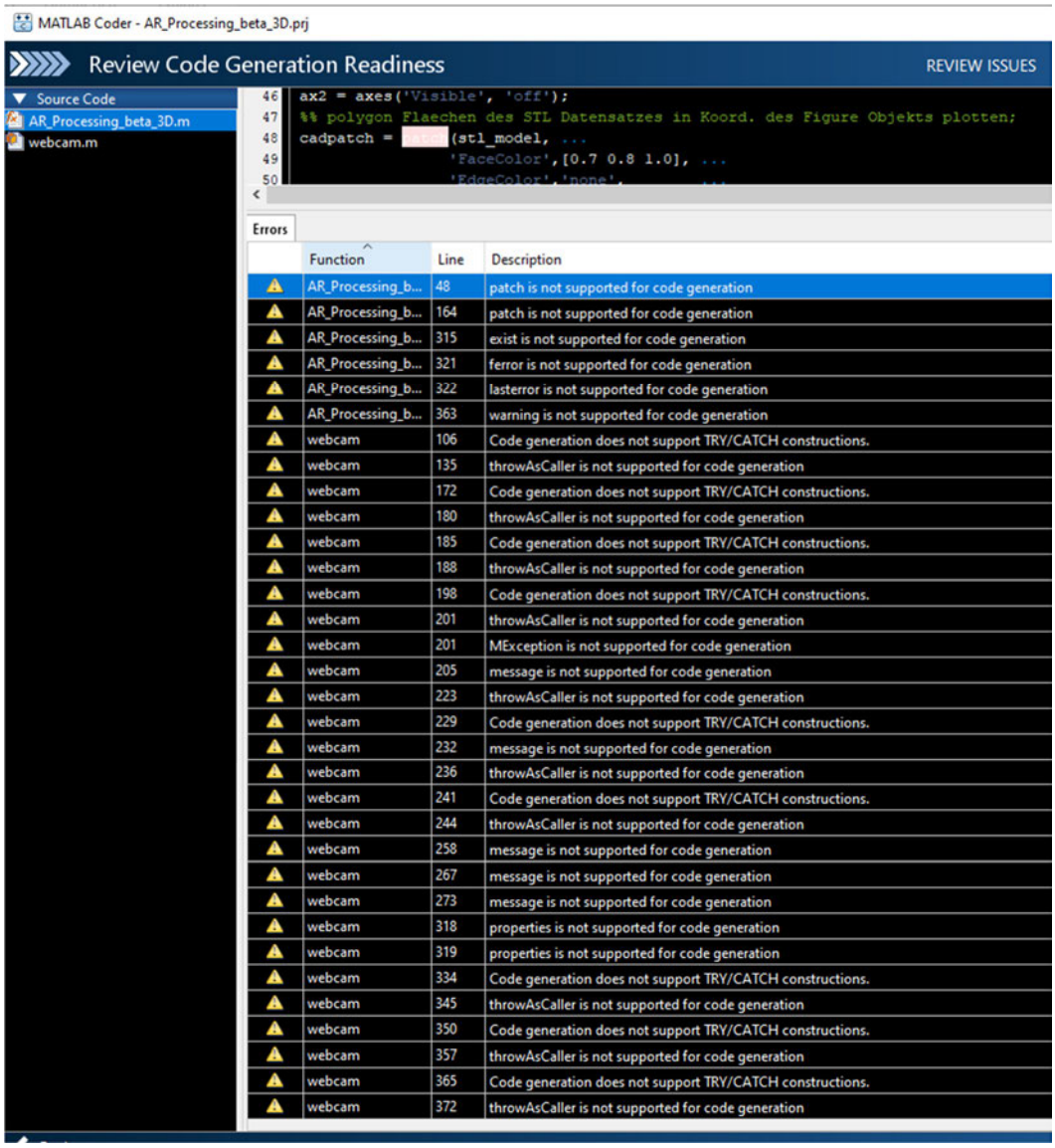


Abbildung 43

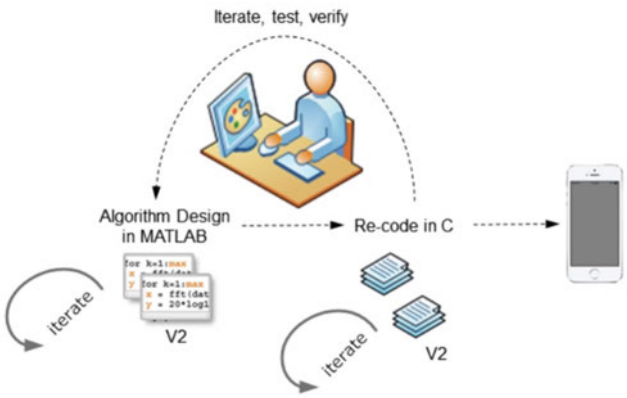


Abbildung 44

Diese Funktion müsste auf niedrigerem Sprachniveau neu geschrieben und passend für das Zielsystem ausgelegt werden. Daher werden bei der Generierung mehrere Iterationen benötigt, bis der Code lauffähig auf einem Tablet, Smartphone o.ä. ist. Schematisch kann man sich die Implementierung als Schleife, wie in Abb. 44 zu sehen, vor Augen führen. Dabei entstehen mit jeder Iterationsschleife Versionen V1-Vn, die sich immer weiter einer voll kompatiblen idealen Smartphone Version annähern. Dementsprechend wird der ursprüngliche Matlab Code oder die fehlerhaft generierte Funktion in C neu codiert. Denkbar sind hierauf aufbauende Ansätze, die sich der vorgestellten Methode widmen, um den Code umzuwandeln.

4 Zusammenfassung und Ausblick

Zum Abschluss dieser Arbeit soll in Anbetracht der ursprünglich aufgestellten Zielsetzung und avisierten Anforderungen eine reflektierende Betrachtung erfolgen und die wesentlichsten Entwicklungsergebnisse festgehalten werden. Darüber hinaus werden, für die offenen Punkte, Möglichkeiten für eine anzustrebende Weiterentwicklung aufgezählt und theoretische Ansätze, die als Anstoß dieser dienen können, nahegebracht.

4.1 Abgleich mit der Zielsetzung und Einschränkungen

Um mit der in 2.3 definierten Zielsetzung zu beginnen lässt sich sagen, dass eine Lösung des Ziels einer einfachen Augmented Reality-Umgebung, zur optischen Überlagerung einer realen Umgebung mit einem CAD Objekt, mit der entwickelten Lösung aus 3.3 darstellbar ist.
Alle funktionalen Anforderungen werden im Grundsatz bedient und realisiert. Es kann in Matlab eine bewegte Aufnahme erzielt werden, die im weiteren Verlauf einsetzbar ist, um eine geeignete Erkennungsmethode zu finden. Die Erkennung wird, mithilfe des SURF Algorithmus, in einem Marker basierten Ansatz umgesetzt. Die Markererkennung funktionierte bereits in der ersten Entwicklungsstufe im zweidimensionalen und konnte auf die 3D Variante adaptiert werden. Zusätzlich ist die Erkennung derart stabil aufgebaut, dass die Laufzeit nur durch manuelles Eingreifen beendet wird oder nach Ende der vorgegebenen Iterationen. So kann bei Start der Anwendung ohne Rücksichtnahme auf eine mögliche Fehlerkennung die Kamera ausgerichtet werden bis genügend matching Punkte gefunden wurden. Daraufhin ist die initiale Position eindeutig definiert und zu jeder Laufzeit zuverlässig nach einer Erkennung. Man kann hier jedoch gegenargumentieren und die Anforderung im Detail damit benennen, dass eine initiale Position auch in Abhängigkeit des Kamerawinkels und der Ausrichtung des Markers erfolgen könnte. Dieser Punkt

würde gegebenenfalls eine ansehnliche Methode bieten, hängt jedoch auch von dem zu erwartenden Anwendungsfall in späteren Stadien einer Weiterentwicklung ab. So könnte man auch wollen, dass das CAD Objekt stets mit einer bestimmten Ebene zur Kamera ausgerichtet ist. Das Tracking folgt allen Bewegungen der Kamera bzw. relativ der des Markers. Einzig die Rotation der Ansicht wird nicht mit dem vorhandenen Endstand des Prototyps verfolgt. In 4.2 wird ein Ansatz zur Realisierung einer vollkommen perspektivischen Ansicht dargelegt. Der vorhandene Stand enthält noch keine Berücksichtigung von großen Distanzänderungen. Beispielsweise müsste sich das CAD Objekt kleiner skalieren, wenn sich der Marker weit von seiner momentanen Distanz zur Kamera entfernt. Ein Konzept für eine Skalierung dieser Art wird ebenfalls in 4.2 kurz erläutert. Andererseits ist in dem Fall noch eine Prüfung der Tragbarkeit dieser Skalierung zu untersuchen, da die Distanz zur Kamera auch ausschlaggebend dafür ist ob die Trackingpunkte weiterhin optisch erfasst werden können oder ob das System an seine technisch limitierte Auflösung stößt. Die Sinnhaftigkeit dürfte je nach vorhandener Hardware und Rechenleistung variieren. Die Berechnung der neuen Positionierung fällt mit den genannten Aspekten einher. Der letzte Stand des Prototyps positioniert das CAD Objekt stets auf dem Marker, dessen absolute Position in einer Ebene ständig neu berechnet wird, nach jeder Bewegung der Kamera. Zur neuen Positionierung fehlt, wie schon beim Tracking erwähnt eine Rotation und damit eine relative Winkeleinstellung der Ausgangsausrichtung des Objekts zu dessen lokalem Koordinatensystem. Der Zugriff auf die CAD Daten wird entsprechend den gestellten Anforderungen umgesetzt und bietet breit aufgestellte Möglichkeiten den Input zu erweitern und den Zugriff auf ein referenziertes Verzeichnis zu dynamisieren, um es neuen Anforderungen anzupassen. Da die Matlab Desktop Version des Prototyps, auf einem Windows Betriebssystem läuft, wird ein standardisierter Datentransfer genutzt, der sich leicht an entfernte Server anknüpfen lässt, sodass ein Zugriff oder eine Speicherung leicht zugänglich ist. Die Echtzeit Überlagerung wird mit der schon vorgestellten Funktion zum Auslesen von STL Datensätzen und der Matlab internen Funktion „patch“ dargestellt. Durch das Auslesen der STL Datei sind nahezu alle Oberflächendaten des Modells vorhanden. Jedoch sind damit auch keine Volumenkörper oder tiefgreifende Informationen enthalten. Man könnte zum Beispiel bei einer Erweiterung des STL Readers auch weitere Absätze der binären Daten auswerten bzw. untersuchen, um einen Transfer der weiteren CAD Daten, wie Parameter, physische Eigenschaften oder Features aus denen sich das Volumenmodell zusammensetzt, zu erreichen. In Kombination mit einer erweiterten Transformation der Ansicht könnte man CAD Bauteile wie in der Modellvorschau im Modellierungsmodus eines CAD Programms, jedoch umgeben von realen Objekten, betrachten. Insgesamt liegt hier ein Prototyp vor, der mit einfachen Mitteln in Matlab realisiert werden kann und einen Einstieg in die Entwicklung von weitreichenderen Augmented Reality-Anwendungen bietet. Wie in 3.4 beschrieben kann man den zugrundeliegenden Prozess als Basis nehmen, um daraus eine Anwendungssoftware für mobile Endgeräte zu erstellen.

Zusammenfassend sorgt die Beachtung aller genannten Aspekte und Bestandteile einer solchen Anwendung dafür, dass eine Vielfalt an informationstechnischen Errungenschaften untersucht und diskutiert werden, die in Teils unterschiedlichen Fachbereichen zum Einsatz kommen können.

4.2 Lösungsansatz für eine Weiterentwicklung

Für die Umsetzung der offenen Punkte wird im nachstehenden ein Ansatz präsentiert, der auf fundamentale Weise bei der Weiterentwicklung helfen kann. Für die perspektivische Ansicht bzw. die Rotation der Objektausrichtung zur Kamera, kann zum Beispiel eine Abwandlung der Transformation, in der 2D Lösung der ersten Entwicklungsstufe, angestrebt werden. Im Wesentlichen handelt es sich hierbei auch um eine Winkel Bestimmung, die rein optisch basiert ist. Dabei wird mit der Funktion „estimateGeometricTransform2D“ eine Transformationsmatrix ausgegeben. Der große Vorteil besteht also darin, dass keine weiteren Sensoren benötigt werden und diese Matrix bereits als Output in der vorhandenen Prototypversion generiert wird. Um zu nachzuvollziehen wie eine Winkeltransformation aus der Perspektive einer viereckigen Fläche gewonnen wird, muss man sich das Prinzip der Transformationsmatrix verdeutlichen. Dieses ist nach den Prinzipien, die in der projektiven Geometrie verwendet werden aufgebaut [88] [Part II: Projective Transformations in 2D | by Daniel Lenton | Medium]. Unter der Projektivität versteht man eine lineare Transformation, die aus der Multiplikation eines 3x1 Vektor mit einer regulären 3x3 Matrix, der Transformationsmatrix, hervorgeht. Eine Projektive Transformationsmatrix hat den größten Freiheitsgrad von allen erdenklichen Matrizen für eine geometrische Transformation. Insgesamt werden mit einem Freiheitsgrad von 8 alle Translatorischen und rotatorischen Transformationen bzw. die verantwortlichen Bewegungen abgebildet. Ebenso kann die in 4.1 genannte Skalierung mit der projektiven Matrix berechnet werden. Somit sind jegliche, räumliche Transformationen auf Basis dieser Methode möglich. Ziel wäre es für eine Weiterentwicklung, die Matrix so zu interpretieren, dass man alle nötigen Transformationswerte extrahiert. Diese wären zum einen die Translationen in Bildkoordinaten, die Winkel Änderung in Objektkoordinaten relativ zu jeder Raumachse und die Skalierung des Objekts bzw. des Objektkoordinatensystems in Abhängigkeit der Distanz zum Marker.

5 Quellenverweise

[1] R. Azuma, „http://www.cs.unc.edu/~azuma/ARpresence.pdf,“ [Online].

[2] blippar, „https://www.blippar.com/blog/2018/08/21/what-is-augmented-reality-and-how-does-augmented-reality-work,“ [Online].

[3] B. K. G. F.-D. T. Moritz Quandt,„https://pdf.sciencedirectassets.com/282173/1-s2.0-S2212827118X00079/1-s2.0-S221282711830163X/main.pdf?X-Amz-Security-Token=IQoJb3JpZ2luX2VjEHQaCXVzLWVhc3QtMSJHMEUCIGX074Nah%2FbTPSoUgmN0681%2F8O2Lhbks1o8qjqb5ONEWAiEAqCZcxAuQ85S8Q#qnNmeu2a6aMjxQdi9v5a%2BUJxN,“ [Online].

[4] „https://docs.opencv.org/3.4/df/d54/tutorial_py_features_meaning.html,“ [Online].

[5] „https://laptrinhx.com/multiple-object-tracking-using-person-re-identification-3225103230/" [Online].

[6] „https://www.e-teaching.org/didaktik/gestaltung/augmented_reality,“ [Online].

[7] S. Li, „https://iopscience.iop.org/article/10.1088/1757-899X/231/1/012003/pdf,“ [Online].

[8] OpenCV, „https://docs.opencv.org/master/d2/d64/tutorial_table_of_content_objdetect.html“ [Online].

[9] W. Qui, „"A vision-based augmented reality system for visualization interaction,"NinthInternational Conference on Information Visualisation (IV'05),2005, pp. 404-409,doi: 10.1109/IV.2005.15.,“ [Online]. Available: https://ieeexplore.ieee.org/document/1509108.

[10] Mathworks Inc., „estimateGeometricTransform2D,“ [Online]. Available: https://de.mathworks.com/help/vision/ref/estimategeometrictransform2d.html?s_tid=doc_ta.

[11] A. Z. P.H.S. Torr, „MLESAC: A New Robust Estimator with Application to Estimating Image Geometry,Computer Vision and Image Understanding,Volume 78, Issue 1,2000,Pages 138-156,ISSN 1077-3142,,“ [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314299908329.

[12] I. &. a. B. I. P. &. P. D. Indrawan, „Indrawan, I & a Bayupati, I Putu & Putri, Desy. (2018). Markerless Augmented Reality Utilizing Gyroscope to Demonstrate the Position of Dewata Nawa Sanga. International Journal of Interactive Mobile Technologies (iJIM). 12. 19. 10.3991/ijim.v12i1.7527.,“ [Online]. Available: https://www.researchgate.net/publication/322666585_Markerless_Augmented_Reality_Utilizing_Gyroscope_to_Demonstrate_the_Position_of_Dewata_Nawa_Sanga.

[13] Mathworks Inc.„Tracking and Motion Estimation,“[Online]. Available:https://de.mathworks.com/help/vision/tracking-and-motion-estimation.html?s_tid=CRUX_lftnav.

[14] Mathworks Inc., „vision.HistogramBasedTracker,“ [Online]. Available: https://de.mathworks.com/help/vision/ref/vision.histogrambasedtracker-system-object.html.

[15] Mathworks Inc., „Tomasi, Carlo and Takeo Kanade. Detection and Tracking of Point Features Computer Science Department, Carnegie Mellon University, April, 1991.,“ [Online]. Available: https://de.mathworks.com/help/vision/ref/vision.pointtracker-system-object.html.

[16] Mathworks Inc., „vision.BlockMatcher,“ [Online]. Available: https://de.mathworks.com/help/vision/ref/vision.blockmatcher-system-object.html.

[17] Mathworks Inc., „vision.BlockMatcher,“ [Online]. Available: https://de.mathworks.com/help/vision/ref/vision.blockmatcher-system-object.html.

[18] Mathworks Inc., „MATLAB Support Package for Apple iOS Sensors,“ [Online]. Available: https://de.mathworks.com/matlabcentral/fileexchange/51235-matlab-support-package-for-apple-ios-sensors.

[19] Mathworks Inc., „patch,“ [Online]. Available: https://de.mathworks.com/help/matlab/ref/patch.html?s_tid=doc_ta.

[20] Mathworks Inc., „mesh,“ [Online]. Available: https://de.mathworks.com/help/matlab/ref/mesh.html.

[21] Core Technologie, „Geometrievereinfachung,“ [Online]. Available: https://www.coretechnologie.de/produkte/3d-evolution/geometrie-vereinfachung.html.

[22] „E.Johnson (2021). STL File Reader(https://www.mathworks.com/matlabcentral/file-exchange/22409-stl-file-reader), MATLABCentral File Exchange.Retrieved July 7, 2021.,“ [Online]. Available:https://de.mathworks.com/matlabcentral/fileexchange/22409-stl-file-reader.

[23] Marxentlabs, „The Ultimate Guide to Markerless Augmented Reality,“ [Online]. Available: https://www.marxentlabs.com/what-is-markerless-augmented-reality-dead-reckoning/.

[24] Apple Inc., „augmented-reality,“ [Online]. Available: https://developer.apple.com/augmented-reality/.

[25] [Online]. Available: https://towardsdatascience.com/5-step-guide-to-generate-3d-meshes-from-point-clouds-with-python-36bad397d8ba.

[26] Mathworks Inc., „Detect SURF features and return SURFPoints object - MATLAB detectSURFFeatures - MathWorks Deutschland,“ [Online]. Available: https://de.mathworks.com/help/vision/ref/detectsurffeatures.html.

[27] T. T. a. L. V. G. Herbert Bay, „SURF: Speeded Up Robust Features,“ [Online]. Available: https://web.archive.org/web/20150319074337/http:/www.vision.ee.ethz.ch/~surf/eccv06.pdf.

[28] M. J. Paul Viola, „Rapid object detection using a boosted cascade of simple features (2001),“ [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.6807.

[29] F. C. Crow, „Summed-Area Tables for Texture Mapping,“ [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.1904&rep=rep1&type=pdf.

[30] S. K. P. S. Jakob Huber, „Analyse von Bildmerkmalen,“ [Online]. Available: https://madoc.bib.uni-mannheim.de/33097/1/Kopf_2013d.pdfAnalyse von Bildmerkmalen.

[31] T. K. Carlo Tomasi, „Detection and Tracking of Point Features,“ [Online]. Available: https://cecas.clemson.edu/~stb/klt/tomasi-kanade-techreport-1991.pdf.

[32] M. Hall, „projective planes,“ [Online]. Available: https://www.ams.org/journals/tran/1943-054-02/S0002-9947-1943-0008892-4/S0002-9947-1943-0008892-4.pdf.

[33] Mathworks Inc., „Matlab Drive,“ [Online]. Available: https://de.mathworks.com/products/matlab-drive.html.

[34] Mathworks Inc., „MATLAB Mobile,“ [Online]. Available: https://de.mathworks.com/products/matlab-mobile.html.

[35] Mathworks Inc., „Image Acquisition Toolbox,“ [Online]. Available:

https://de.mathworks.com/products/image-acquisition.html.

[36] Mathworks Inc., „FInding matching features - MATLAB matchFeatures,“ [Online]. Available: https://de.mathworks.com/help/vision/ref/matchfeatures.html?s_tid=doc_ta.

[37] Mathworks Inc., „2-D projective geometric transformation - MATLAB,“ [Online]. Available: https://de.mathworks.com/help/images/ref/projective2d.html.

[38] Mathworks Inc., „2-D affine geometric Transformation,“ [Online]. Available: https://de.mathworks.com/help/images/ref/affine2d.html.

[39] Mathworks Inc., „What Are System Objects? - MATLAB & Simulink - MathWorks Deutschland,“ [Online]. Available: https://de.mathworks.com/help/matlab/matlab_prog/what-are-system-objects.html.

[40] Mathworks Inc., „MATLAB to iPhone and Android Made Easy,“ [Online]. Available: https://de.mathworks.com/videos/matlab-to-iphone-and-android-made-easy-107779.html.

[41] Porsche, „https://www.youtube.com/watch?v=zx0Y1UBvhgk&t=3s,“ [Online].

[42] „Getting Started with Google ARCore, Part 2: Visualizing Planes & Placing Objects,“ [Online]. Available: https://www.andreasjakl.com/getting-started-with-google-arcore-part-2-visualizing-planes-placing-objects/.

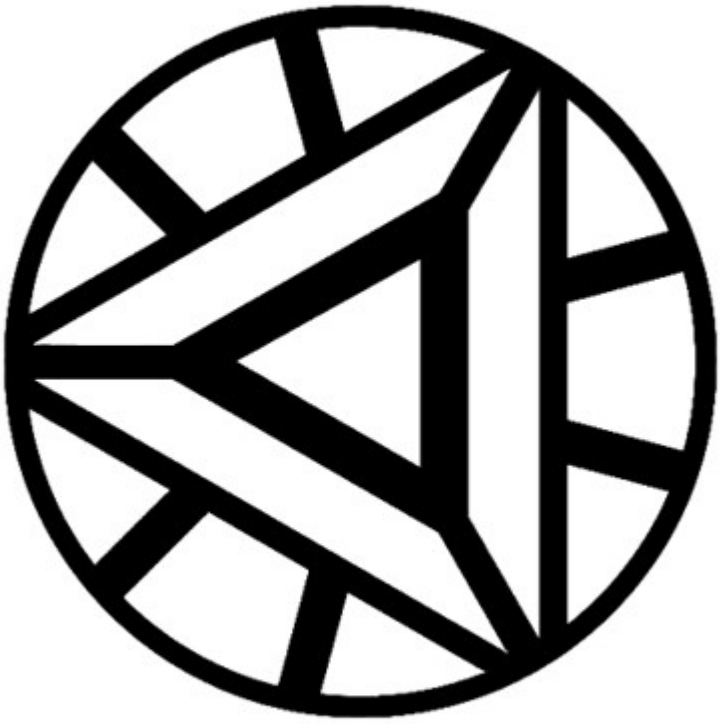
[43] „https://www.researchgate.net/figure/Occluding-edges-in-geometric-images-with-bistatic-sensor_fig6_220566232,“ [Online]. Available: https://www.researchgate.net/figure/Occluding-edges-in-geometric-images-with-bistatic-sensor_fig6_220566232.

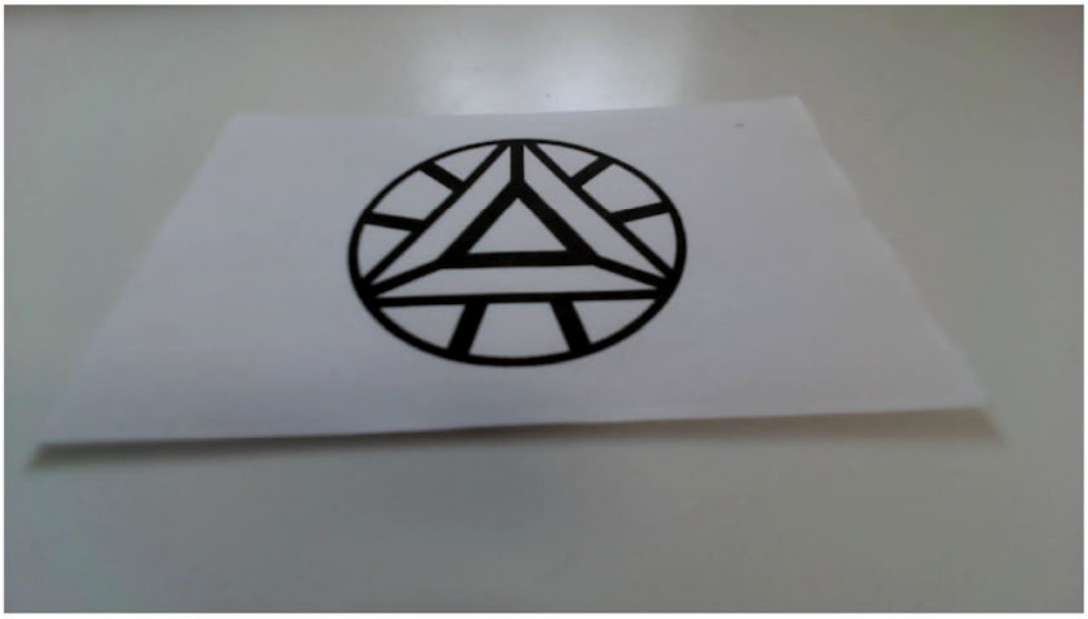
[44] J.-C. &. K. S.-D. (. Piao, „Adaptive Monocular Visual–Inertial SLAM for Real-Time Augmented Reality Applications in Mobile Devices - Scientific Figure on ResearchGate.,“ [Online]. Available: https://www.researchgate.net/figure/Diagram-of-the-Kanade-Lucas-Tomasi-KLT-feature-tracking-algorithm_fig7_320916069.

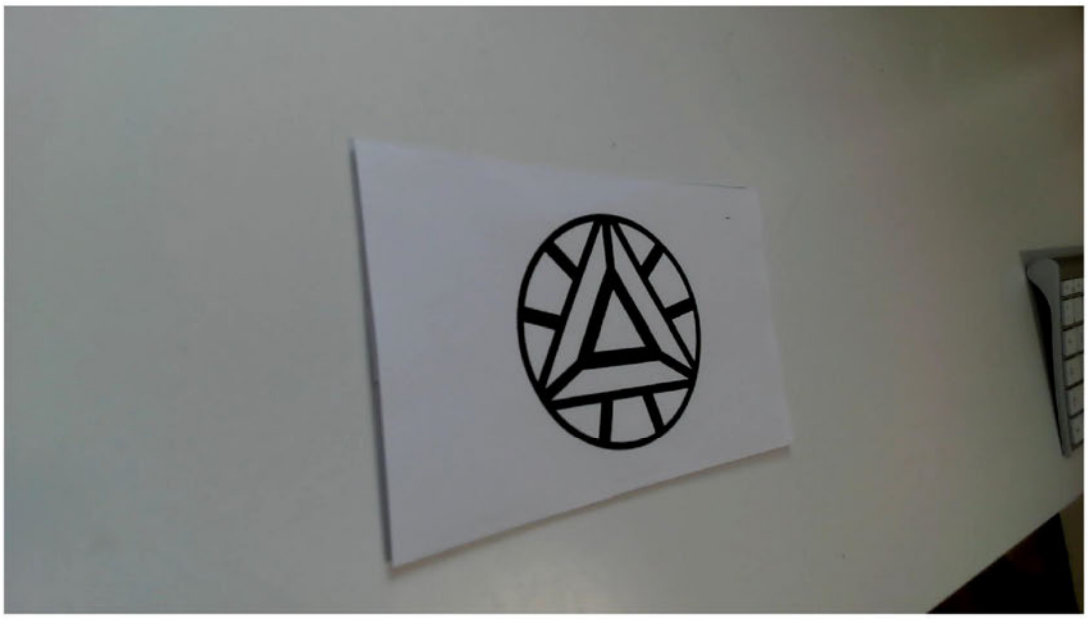
Anhang A: Testreihe zu Objekterkennungsalgorithmen in Matlab

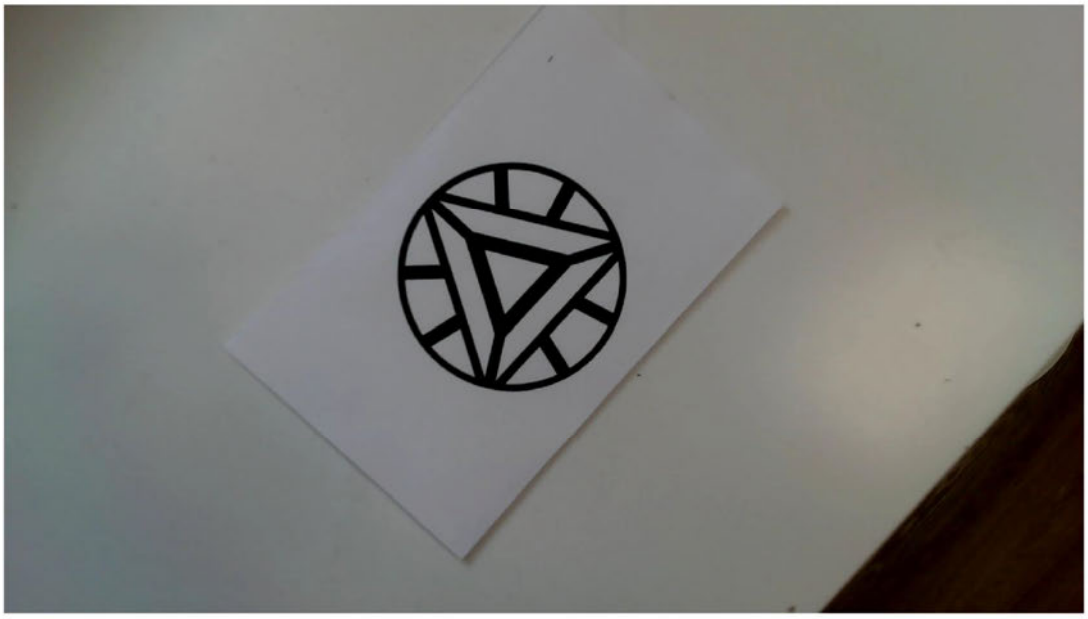
Hinweis: Alle Anhänge (A-D) sind als vollständiges Verzeichnis in der CDROM beigelegt.

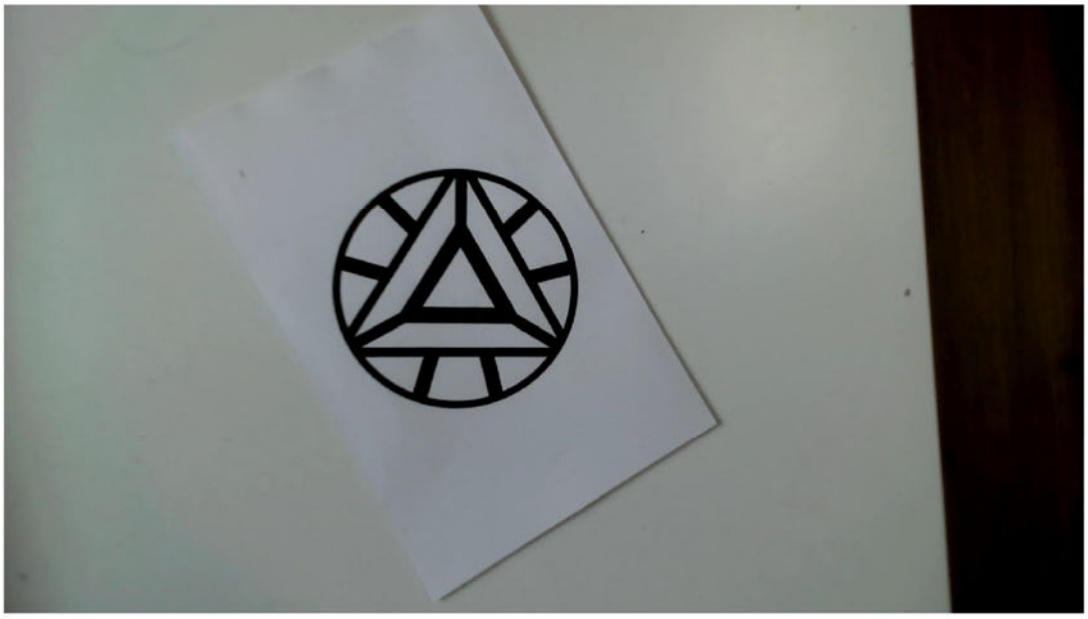
Die Matlab Skripte müssen bitte jeweils mit allen anderen im jeweiligen Verzeichnis befindlichen Dateien in Matlab geladen werden (Zugriff durch Matlab auf den Speicherort wird vorausgesetzt).











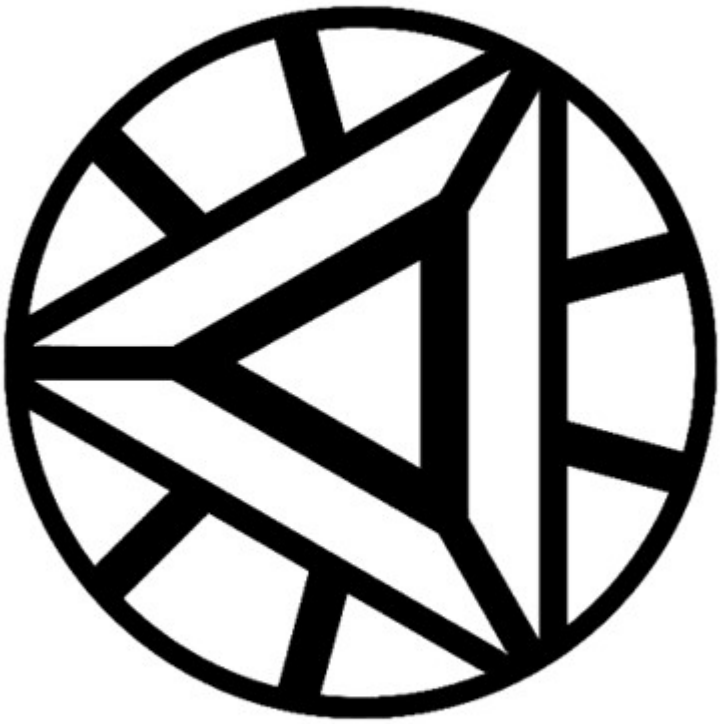
test_detection.m – Auswertungsskript in Matlab

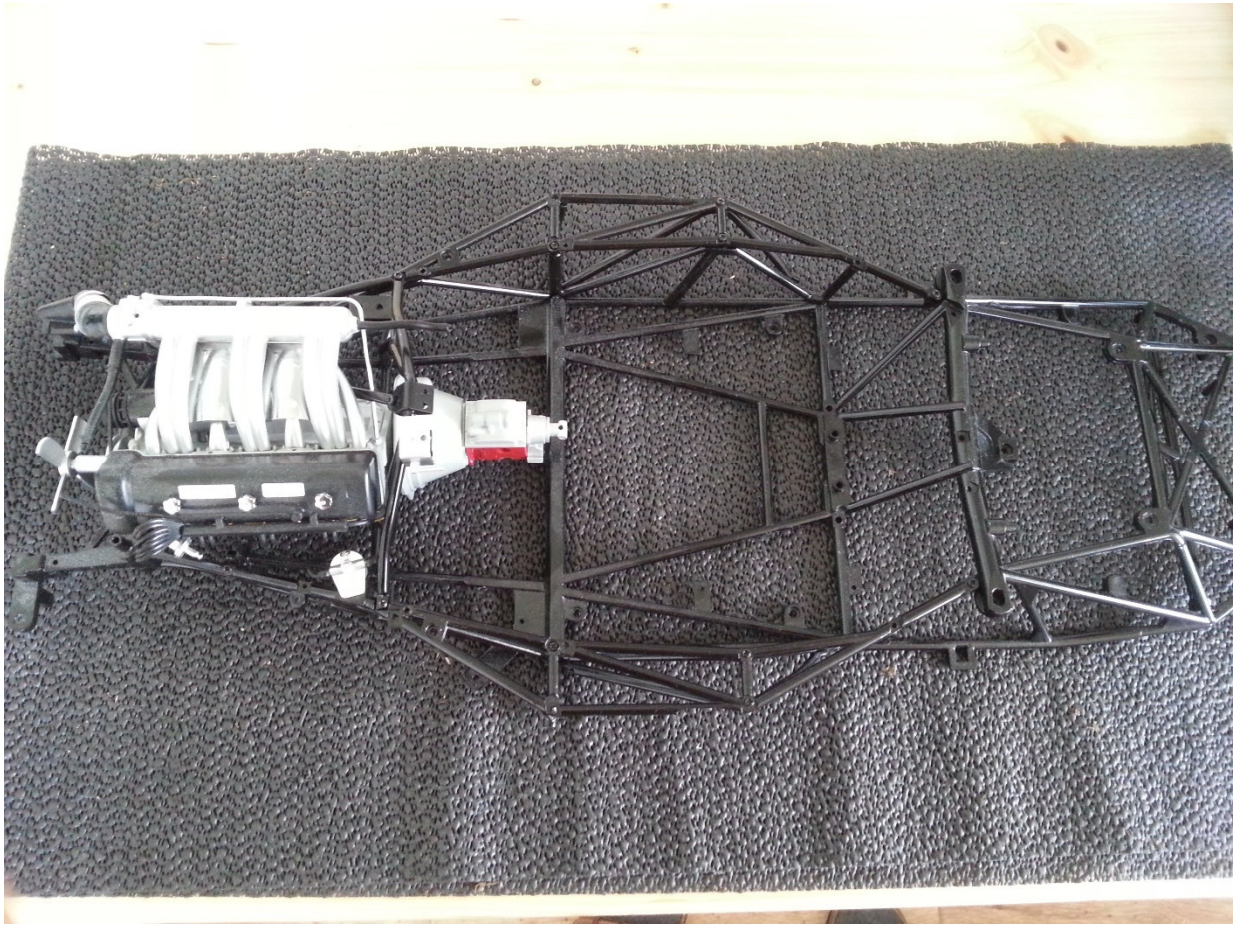
```
%Testreihe: Objekterkennungsalgorithmen
%take_pics;
clc
clear all;
close all;
%% test_SURF()
for i=1:6
    SURF.t1=now;
    SURFObj = detectSURFFeatures(rgb2gray(im-
read("Frame"+num2str(i)+".jpg")));
    SURF.t2=now;
    SURF.points_count(i) = SURFObj.Count;
    SURF.delta_t(i)=SURF.t2-SURF.t1;
end
%durschn. zeit
```

```
SURF.avg_delta_t = mean(SURF.delta_t,2)* 24 * 60 * 60;
%durchschn. anz.
SURF.avg_points_count = mean(SURF.points_count);
%stdabw. Rotation
SURF.std_rotation = std(SURF.points_count(4:6));
%....analog für alle weiteren Funktionen
%% test_FAST()
 for i=1:6
    FAST.t1=now;
    FASTObj = detectFASTFeatures(rgb2gray(im-
read("Frame"+num2str(i)+".jpg")));
    FAST.t2=now;
    FAST.points_count(i) = FASTObj.Count;
    FAST.delta_t(i)=FAST.t2-FAST.t1;
 end

FAST.avg_delta_t = mean(FAST.delta_t,2)* 24 * 60 * 60;
FAST.avg_points_count = mean(FAST.points_count);
FAST.std_rotation = std(FAST.points_count(4:6));
%% test_ORB()
for i=1:6
    ORB.t1=now;
    ORBObj = detectORBFeatures(rgb2gray(imread("Frame"+num2str(i)+".jpg")));
    ORB.t2=now;
    ORB.points_count(i) = ORBObj.Count;
    ORB.delta_t(i)=ORB.t2-ORB.t1;
end
ORB.avg_delta_t = mean(ORB.delta_t,2)* 24 * 60 * 60;
ORB.avg_points_count = mean(ORB.points_count);
ORB.std_rotation = std(ORB.points_count(4:6));
%% test_BRISK()
for i=1:6
    BRISK.t1 = now;
    BRISKObj = detectBRISKFeatures(rgb2gray(im-
read("Frame"+num2str(i)+".jpg")));
    BRISK.t2 = now;
    BRISK.points_count(i) = BRISKObj.Count;
    BRISK.delta_t(i)=BRISK.t2-BRISK.t1;
end

BRISK.avg_delta_t = mean(BRISK.delta_t,2)* 24 * 60 * 60;
BRISK.avg_points_count = mean(BRISK.points_count);
BRISK.std_rotation = std(BRISK.points_count(4:6));
%% benchmark Werte
benchmark.count = [SURF.avg_points_count...
                   ORB.avg_points_count...
                   FAST.avg_points_count...
                   BRISK.avg_points_count...
                   ];
benchmark.delta_t = [SURF.avg_delta_t...
                   ORB.avg_delta_t...
                   FAST.avg_delta_t...
                   BRISK.avg_delta_t...
                   ];
benchmark.std_rotation = [SURF.std_rotation...
                   ORB.std_rotation...
                   FAST.std_rotation...
                   BRISK.std_rotation...
                   ];

xaxes_names = categorical({'SURF','ORB','FAST','BRISK'});
xaxes_names = reordercats(xaxes_names,{'SURF','ORB','FAST','BRISK'});
```

```
%% plot der Balkendiagramme
figure("Name","Anzahl der erkannten Punkte");
b_count = bar(xaxes_names,benchmark.count);
title(strcat('$D_{n}$ ',get(gcf,'Name')),'interpreter','latex');
xtips1 = b_count.XEndPoints;
ytips1 = b_count.YEndPoints;
labels1 = string(b_count.YData);
ylabel('Anzahl der Merkmalspunkte $n$','interpreter','latex');
text(xtips1,ytips1,labels1,'HorizontalAlignment','center',...
    'VerticalAlignment','bottom');
saveas(gcf,strcat(get(gcf,'Name'),'.png'));
saveas(gcf,strcat(get(gcf,'Name'),'.fig'));
%%
figure("Name","mittlere Laufzeit (in s)");
b_delta_t = bar(xaxes_names,benchmark.delta_t);
title(strcat('$D_{t}$ ',get(gcf,'Name')),'interpreter','latex');
xtips1 = b_delta_t.XEndPoints;
ytips1 = b_delta_t.YEndPoints;
labels1 = string(b_delta_t.YData);
ylabel('Zeit in Sekunden $s$','interpreter','latex');
text(xtips1,ytips1,labels1,'HorizontalAlignment','center',...
    'VerticalAlignment','bottom');
saveas(gcf,strcat(get(gcf,'Name'),'.png'));
saveas(gcf,strcat(get(gcf,'Name'),'.fig'));
%%
figure("Name","Rotationsinvarianz Standardabw. ueber \theta (3-Winkelaende-
rungen)");
b_std_rotation = bar(xaxes_names,benchmark.std_rotation);
title('$D \sigma _{n} Rotationsinvarianz-Standardabw.-(3-Winkelaenderun-
gen)$','interpreter','latex');
xtips1 = b_std_rotation.XEndPoints;
ytips1 = b_std_rotation.YEndPoints;
labels1 = string(b_std_rotation.YData);
ylabel('Standardabweichung $\sigma _{n} $','interpreter','latex');
text(xtips1,ytips1,labels1,'HorizontalAlignment','center',...
    'VerticalAlignment','bottom');
saveas(gcf,'Rotationsinvarianz.png');
saveas(gcf,'Rotationsinvarianz.fig');
%% take_pics
function take_pics()
clear cam;
cam = webcam;
preview(cam);
for i=0:5
    img = snapshot(cam);
    obj = imshow(img);
    saveas(obj,"FrameMusterVgl"+num2str(i)+".jpg");
end
end
```

Anhang B: Augmented Reality Umgebung für die Optische Überlagerung von 2D Bilddaten





```
%Pre-Prozessor
%Kamerakalibrierung
%Merkmalserkennung
%3D Vision and Stero Vision
%Object Tracking unf Bewegungschätzung
%OpenCV Schnittstelle?
%point cloud processing
%Room mapping
clc
close all;
clear;
%%init. d. Bilddaten und Erfassung der markanten Bildeigenschaften
if computer=='PCWIN64'
    mobile = false;
else
    mobile = true;
end
waitfor(msgbox(strcat('Erster Prototyp: Bitte die Kamera auf den Marker
richten.',...
                ' (HINWEIS: Es ist auch eine Erkennung ohne sofortige ',...
                'Ausrichtung möglich jedoch kann es zu Fehlerkennungen kom-
men.)',...
                ' Die Anwendung schließt sich nach einer Weile von
selbst...')));
%% matching der features d. referenzbilddaten mit den kamerabilddaten
firstrun = 1;
%while schleife für Minimumkriterium der Übereinstimmenden Merkmale
%für die Übergabe an das Tracking  -> min. 7 Punkte
while firstrun == 1 | inlierCameraPts.Count < 8
    firstrun = 0;
    if mobile==false
```

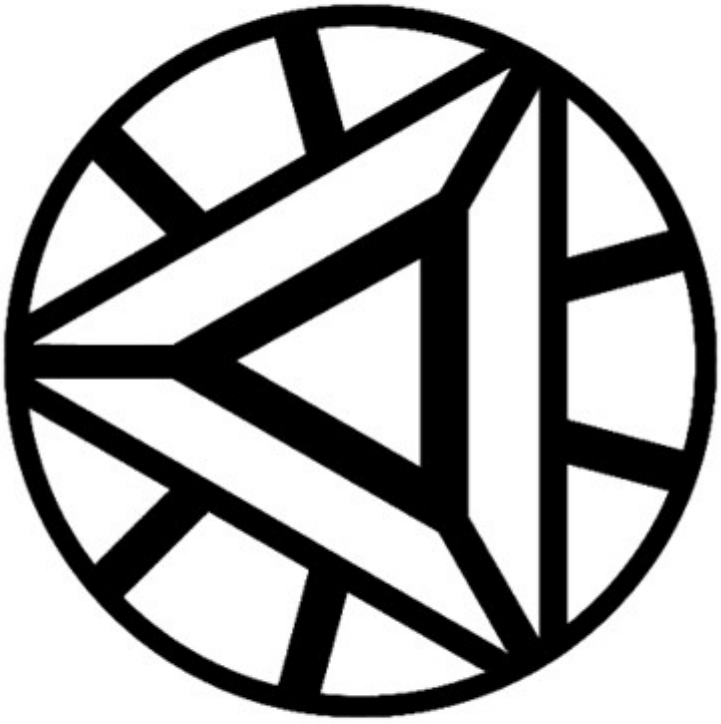
```
        clear inlierCameraPts;
        [replacemedia,refimg,refpts,reffeatures,cam,...
        currentframe,campts,camFeatures,validpts]= ini_imgsource;
    else
        clear inlierCameraPts;
        [replacemedia,refimg,refpts,reffeatures,cam,...
        currentframe,campts,camFeatures,validpts]= ini_imgsource_mobile;
    end
    idxpairs = ...
    matchFeatures(camFeatures, reffeatures,'MaxRatio',0.9);%,...
    %'Method','Approximate','MatchThreshold',15);
    % speichern der jwl. matched surf pkt.
    matchedcampts = campts(idxpairs(:,1));
    matchedrefpts = refpts(idxpairs(:,2));

    figure;
    showMatchedFeatures(...
         currentframe, refimg, matchedcampts, matchedrefpts,'Montage');
    % Transformation zwischen den Referenzbilddaten und den vorliegenden
    % Bilddaten bestimmen
    [refTransform, inlierindx, status] = ...
        estimateGeometricTransform2D(...
              matchedrefpts, matchedcampts,'projective');%'similarity');
    inlierCameraPts = matchedcampts(inlierindx,:);
    inlierReferencePts = matchedrefpts(inlierindx,:);
end
% Transf.vekt.
figure;
showMatchedFeatures(...
     currentframe, refimg, inlierCameraPts, inlierReferencePts,'Montage');
% Darst. Trackingpkt.
%% initialisiere Punkt Tracker S
close all;
pointtracker = vision.PointTracker('MaxBidirectionalError',3,...
                                   'MaxIterations',50);
initialize(pointtracker, inlierCameraPts.Location, currentframe);
trackingmarkers = insertMarker(currentframe, inlierCameraPts.Location,...
                  'Size',7,'Color','yellow');
figure;
imshow(trackingmarkers);
%% redimensionierung der ersetzenden Mediendaten
% lade Austauschmedium
frame = replacemedia;
% get replace and ref dims
repdims = size(frame(:,:,1));
refdims = size(refimg);
%Transformationskalierung ermitteln die das Austauschmedium auf die
%aufgenommene Bildgröße/Markergröße skaliert
%preserving aspect ratio
scaletransform = estimateScale(refdims,repdims);
outputview = imref2d(size(refimg));
%vidframescaled = imwarp(vidframe, scaletransform,'OutputView', outputview);
vidframescaled = imwarp(frame, scaletransform,'OutputView', outputview);
figure;
imshowpair(refimg, vidframescaled,'Montage');
%% transformation auf die ersetzende Mediendaten anwenden
outputview =imref2d(size(currentframe));
vidframetransformed = imwarp(vidframescaled,refTransform, ...
                             'OutputView',outputview);
figure;
imshowpair(currentframe, vidframetransformed,'Montage');
%% Einfügen/Überlagern des transformierten Austauschbildes ins Kamerabild
```

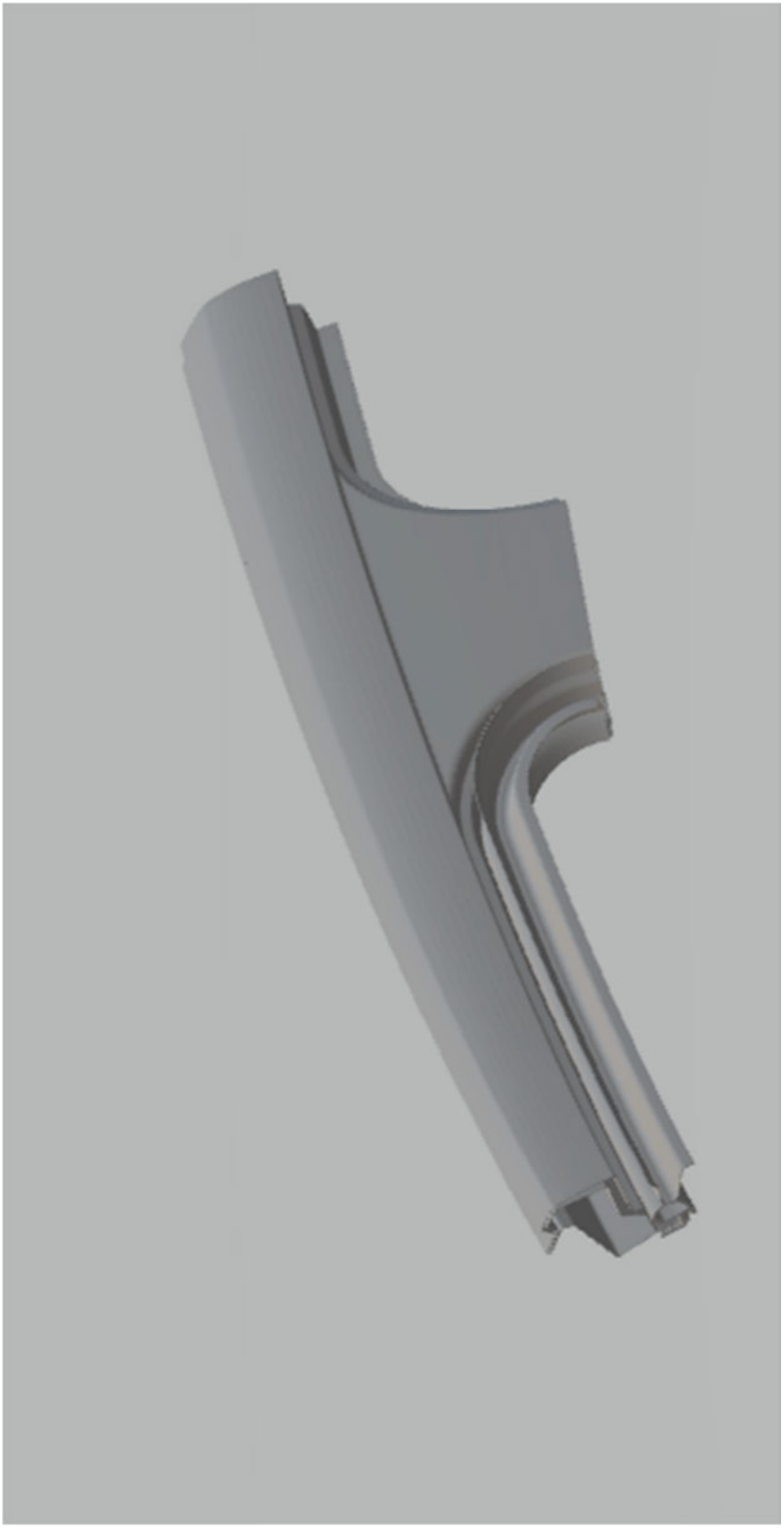
```
alphablender = vision.AlphaBlender(...
    'Operation','Binary mask','MaskSource','Input port');
mask = vidframetransformed(:,:,1) | ...
       vidframetransformed(:,:,2) | ...
       vidframetransformed(:,:,3) > 0;
outputframe = step(alphablender, currentframe, vidframetransformed, mask);
figure;
imshow(outputframe);

%% track Pkt auf nächshtes Bild
    isValid = 0;
    counter = 0;
    if mobile==true
        clear cam
        dev = mobiledev;
        cam = camera(dev,'back');
        cam.Autofocus = 'on'
        cam.Resolution = '1280x720';
    end
    while counter <= 100
        %% naechstes Kamerabild
             %(nnz(isValid)<=2) | counter <= 10
            % muss mind. 2 traking points zw. den 2 Bildern
            prevcurrentframe = currentframe;
            scanningloop = 0;
            %while (nnz(isValid)<=2) %| scanningloop <= 10
            %pause(0.1);


            if mobile==false
                currentframe = snapshot(cam);
            else
                currentframe = snapshot(cam,'immediate');
            end
            % neue, gültige tracking pkt. finden
            [trackedpoints, isValid] = step(pointtracker, currentframe);
            % nur die Pkt übernehmen die exakt getracked wurden
            newvalidlocs = trackedpoints(isValid,:);
            oldvalidlocs = inlierCameraPts.Location(isValid,:);

            clear isValid;
        %% Ermittlung d. geometr. Transformation zw. Zwei Bildern
            if size(newvalidlocs,1)>4
            [trackingtransform, inldx_oldvnew]=...
                        estimateGeometricTransform2D(...
                        oldvalidlocs, newvalidlocs,'projective');% 'simila-
rity');

%             oldinlierloc = oldvalidlocs(inldx_oldvnew,:);
%             newinlierloc = newvalidlocs(inldx_oldvnew,:);
            %     figure(1);
            %     showMatchedFeatures(prevcurrentframe,currentframe,...
            %             oldinlierloc,newinlierloc,'Montage');
            % reset Point tracker for next frame tracking
            setPoints(pointtracker, newvalidlocs);
        %% Anwendung d. geomet. transf. von 'refTransform' auf 'current-
frame'
            trackingtransform.T = refTransform.T * trackingtransform.T;
            % erneute redimensionierung der ersetzenden Bilddaten
            %repframe = step(vid);
            outputview = imref2d(size(refimg));
            %    vidframescaled = imwarp(repframe, scaletransform, ...
```

```
            %                                   'OutputView',outputview);
             vidframescaled = imwarp(frame, scaletransform, ...
                                     'OutputView',outputview);
            %   figure(1);
            %   imshowpair(refimg,vidframescaled,'Montage');
                % erneute Transformation der ersetzenden redimensionierten
Bilddaten
            outputview = imref2d(size(currentframe));
            vidframetransformed = imwarp(vidframescaled,trackingtransform,
...
                                     'OutputView',outputview);
            %   figure(1);
            %   imshowpair(refimg, vidframescaled,'Montage');
                % insert transformed replace vid frame into webcam frame
            mask = vidframetransformed(:,:,1) | ...
                   vidframetransformed(:,:,2) | ...
                   vidframetransformed(:,:,3) > 0;
          end
        %% Darstellung der naechsten Aufnahme
            outputframe = ...
                step(alphablender, currentframe, vidframetransformed, mask);
            figure(1);
            imshow(outputframe);

            counter = counter + 1;
    end
%% ========================================================================
% reset: Verbindung zur Hardware trennen und Speicher leeren
clear all;
%% ========================================================================
% functions:
    %% Skalierung fuer die Bildtransofrmation ermitteln
    function [scaleTransform ]=estimateScale(refDims,repDims)
        if (repDims(1)/repDims(2) > 1) && (refDims(2)/refDims(2) > 1)
            % Wenn das Verhaeltnis zwischen Laenge und Breite gleich ist
            scaleTransform=affine2d([1/(repDims(1)/refDims(1)) 0 0 ; ...
                            0 1/(repDims(2)/refDims(2)) 0 ; 0 0 1]);
        else
            % Wenn das Verhaeltnis zwischen Laenge und Breite unterschied-
lich ist
            scaleTransform=affine2d([0 1/(repDims(2)/refDims(1)) 0 ; ...
                            1/(repDims(1)/refDims(2)) 0 0 ; 0 0 1]);
        end
    end
    %% Initialisierung der Bilddaten;Erkennung von SURF Punkten
    % Extrahieren der features für die Übergabe an matchFeatures
    function [replacemedia,refimg,refpts,reffeatures,cam,currentframe,...
                campts,camFeatures,validpts] = ini_imgsource()
        % Initi. Referenzbild 'refimg' und Bildmerkmale erfassen
            refimg = imread('firstimg.jpg');
            %refimg = imread('m4.jpg');
            % erkennen + extrahieren der 'SURF Features'
            refimggray = rgb2gray(refimg);
            refpts = detectSURFFeatures(refimggray);
            reffeatures = extractFeatures(refimggray, refpts);
            % Darstellung der 'SURF Features'
            %figure;
            %imshow(refimg),hold on;
            %plot(refpts.selectStrongest(100));
            %figure;

        %init. Austauschmedium 'replacemedia'
```

```
            replacemedia = imread('sample.jpg');

        % init. kamera 'cam' und Bildmerkmale erfassen
            % verbindung zur Kamera aufbauen
            cam = webcam;
            % erkennen + extrahieren der 'SURF Features' in aktueller Auf-
nahme
            for a=1:10
                currentframe = snapshot(cam);
                currentframegray = rgb2gray(currentframe);
                campts = detectSURFFeatures(currentframegray);%,'NumOcta-
ves',4,'NumScaleLevels',6);
                % darstellung der akt. Aufnahme
                imshow(currentframe);
                hold on;
                [camFeatures, validpts] = extractFeatures(currentframegray,
campts);
                plot(validpts.selectStrongest(75));
            end

    end
    %% Initialisierung der Bilddaten;Erkenn. v. SURF Pkt. (MATLAB mobile)
    function [replacemedia,refimg,refpts,reffeatures,cam,currentframe,...
                campts,camFeatures,validpts] = ini_imgsource_mobile()
        % init. replace video
        replacemedia = imread('sample.jpg');
        % Initi. u. Erfassung v. Bilddaten: 'cam', 'refimg'; Ermittl. SURF
Pkt.
            % load reference img, detect surf points, extract desc
        % init. referenzbild u. erkenne SURF pkt.
        refimg = imread('firstimg.jpg');

            %detect + extract SURF features
        refimggray = rgb2gray(refimg);
        refpts = detectSURFFeatures(refimggray);

        reffeatures = extractFeatures(refimggray, refpts);
            % disp surf features for currentframe
        figure;
    %     imshow(refimg),hold on;
        plot(refpts.selectStrongest(100));
        % init. kamera und erkenne SURF pkt.
            % Connect to selected webcam
        dev = mobiledev;
        cam = camera(dev,'back');
        cam.Autofocus = 'on'
        cam.Resolution = '1280x720';
            % surface detection in frame
        figure;
        for a=1:10
        currentframe = snapshot(cam,'immediate');

        currentframegray = rgb2gray(currentframe);
        campts = detectSURFFeatures(currentframegray);
            % darstellung der akt. Aufnahme
        imshow(currentframe);
        hold on;
        [camFeatures, validpts] = extractFeatures(currentframegray, campts);
        plot(validpts.selectStrongest(100));
        end
    end
%% ========================================================================
```

Anhang C: Augmented Reality Umgebung für die Optische Überlagerung von CAD-Daten





Die Datei des STL Modells ist in der beigelegten CDROM enthalten.

```
%BESCHREIBUNG ERGAENZEN
waitfor(msgbox(strcat('Zweiter Prototyp: ',...
    'Die Kamera kann frei im Raum bewegt werden. ',...
    'Der Marker wird kontinuierlich erkannt.',...
    'Diese Anwendung schließt sich nach einer Weile automatisch')));
clc
close all;
clear variables;
%%
cam = webcam;
%% CAD Daten importieren: Ausgewaehlten STL Datensatz laden
stl_model = ...
    stlread('Moehler_KK3_Hausarbeit_Dachrahmen.stl');
%% initialize point tracker
pointtracker = vision.PointTracker('MaxBidirectionalError',3,...
                                   'MaxIterations',50);
%% Dimensionen der Kameraufloesung speichern
% (fuer darauffolgende dimensionierungen)
x_cam_res = str2double(extractBefore(cam.Resolution,'x'));
y_cam_res = str2double(extractAfter(cam.Resolution,'x'));
%%  matching der features d. referenzbilddaten mit den kamerabilddaten
currentvalidlocs = 0;
firstrun = 1;
%% Figure Objekt 'ar_plot' generieren und Festlegung der Eigenschaften
ar_plot = figure('Name','AR_processing_Mk2_3D',...
                 'NumberTitle','off',          ...
                 'MenuBar','none',             ...
                 'ToolBar','none');
ar_plot.WindowState = 'maximized';
ax1 = axes();
%% inititalisierung des SURF Detektors
[mean_middlepoint, currentvalidlocs, ...
 currentframe, refimg, validpts, ...
 camFeatures, validrefpts,reffeatures] = ...
 initial_detection(firstrun, currentvalidlocs, pointtracker, cam, ax1);
% Mittelpunkt speichern
middlepoint = ...
    [mean_middlepoint(1,1)/x_cam_res ...
     mean_middlepoint(1,2)/y_cam_res];
%axis image:
%Def.: "Sets DataAspectRatio to [1 1 1] and sets the associated mode
%       property to manual"
%skaliert die Achsen proportional zur Aufnahme
axis image;
hold on;
ax2 = axes('Visible', 'off');
%% polygon Flaechen des STL Datensatzes in Koord. des Figure Objekts plot-
ten;
cadpatch = patch(stl_model, ...
                 'FaceColor',[0.7 0.8 1.0], ...
                 'EdgeColor','none',        ...
                 'FaceLighting','gouraud',  ...
                 'AmbientStrength', 0.15,   ...
                 'Parent', ax2);
%axis vis3d matlab def.: stellt den Modus des plot-Raum Seitenverhaeltnis
%und des Daten Seitenverhaeltnis auf manuell -> inidviduell definierbare
%Raum Dimensionierung ermoeglicht 3D Effekte
axis vis3d;
axis image;
camlight('headlight');
camproj('perspective');
ax2.Position = [middlepoint(1,1)-0.34 middlepoint(1,2)-0.17 0.77 0.34];
%origpos = [0,0,0];
```

```
%campos(origpos);
%campos('manual');
% %% Darstellung der Trackingpunkte
% trackingmarkers = insertMarker(currentframe, matchedcampts.Location,...
%                   'Size',7,'Color','yellow');
% figure(ar_plot);
% imshow(trackingmarkers,'Border','tight');
%% merkmalspunkte auf naechstes Bild tracken
   % Iteration fuer eine festgelegte Anzahl an Durchlaeufen
   % Das Tracking der Punkte erfolgt nach jeder erneuten Aufnahme
   % Der komplette Durchlauf erzeugt ein Bewegtes Bild und ermittelt durch-
gehend
   % die Transformation der Trackingpunkte
   % default Werte
   isValid = 0;
   counter = 0;
   azimuth = 0;
   elevation = 45;
   toofew = 0;

while counter <= 250
    % Initialisierung fuer die erste Erkennung
    % (Schleife zur Aufrechterhaltung bis der Marker erneut gefunden wird)
    while sum(isValid) < 4
        patchdelete = 0;
        while size(currentvalidlocs,1) < 4
            if firstrun == 1
                firstrun = 0;
            else
                patchdelete = 1;
                delete(cadpatch);
            end
            [mean_middlepoint, currentvalidlocs, currentframe] = ...
                initial_detection( ...
                    firstrun, currentvalidlocs, pointtracker, cam, ax1);
        end
        % speichern des vorangegangenen Kamerabildes
        prevcurrentframe = currentframe;
        % vorangegangenen Durchschnittsmittelpunkt speichern
        prevmiddlepos = mean_middlepoint;
        prevmiddlepos = [
                         prevmiddlepos(1,1)/x_cam_res-0.34 ...
                         prevmiddlepos(1,2)/y_cam_res-0.17
                        ];
        % naechstes Kamerabild
        currentframe = snapshot(cam);
        % neue, gültige tracking pkt. finden
        [trackedpoints, isValid] = step(pointtracker, currentframe);
        % vorangegange tracking pkt. speichern
        prevvalidlocs = currentvalidlocs(isValid,:);
        % nur die neuen Trackingpkt. übernehmen die exakt getracked wurden
        currentvalidlocs = trackedpoints(isValid,:);
    end

    % neuen Durchschnittsmittelpunkt bestimmen
    mean_middlepoint = mean(currentvalidlocs, 1);
    % Ermittlung der Transformationsmatrix zwischen Zwei Trackingkoordinaten
    [trackingtransform, inlierindx_oldtonew]=...
                estimateGeometricTransform2D(...
                prevvalidlocs, currentvalidlocs,'projective');% 'similari-
ty');
    %disp(trackingtransform.T);
```

```
    oldinlierloc = prevvalidlocs(inlierindx_oldtonew,:);
    newinlierloc = currentvalidlocs(inlierindx_oldtonew,:);
%               showMatchedFeatures(prevcurrentframe,currentframe,...
%                                   oldinlierloc,newinlierloc,'Montage');

    %% Transformation der Ansicht bestimmen
    % Translation:
        % Bewegungsmatrix (m x n: m = Anzahl der getrackten Pkt; n = 2 (x,y)
)
        trackedpoints_translatorymotion = ...
                  trackedpoints(1:size(prevvalidlocs,1),:) - prevvalidlocs;
        % arithmetisches Mittel entlang der ersten Matrix Dimension
        % m x n Matrix zu einem 1 x n Vektor
        % -> durschnittliche translatorische Bewegung der Pkt.(Bildkoordina-
ten)
        trackedpoints_translatorymotion = ...
            double(mean(trackedpoints_translatorymotion,1));
        % Anpassung der Skalierung
        % (basierend auf der Kameraauflösung Koordinaten des Bildes)
        trackedpoints_translatorymotion(1,1) = ...
            trackedpoints_translatorymotion(1,1) / x_cam_res;
        trackedpoints_translatorymotion(1,2) = ...
            trackedpoints_translatorymotion(1,2) / y_cam_res;

    %Rotation und Skalierung:
    % ...
    % ...

    %% Transformation der Ansicht
    % Anwendung des Translationsvektors
    % (Koordinatensystem des CAD Modells neu ausrichten)
    if prevvalidlocs ~= 0
        newviewpos = prevmiddlepos + ... %campos + ...
                     [
                     trackedpoints_translatorymotion(:,1) ...
                     (trackedpoints_translatorymotion(:,2))...
                     ];
        figure(ar_plot);
        delta_y = newviewpos - prevmiddlepos;
        if delta_y == trackedpoints_translatorymotion(:,2)*-1
                      newviewpos = prevmiddlepos + ... %campos + ...
                      [
                      trackedpoints_translatorymotion(:,1) ...
                      (-1*trackedpoints_translatorymotion(:,2))...
                      ];
        end

        ax2.Position = [
                        newviewpos ...
                        0.75 ...
                        0.75 ...
                       ];

       if patchdelete == 1
       cadpatch = patch(stl_model, ...
                        'FaceColor',[0.7 0.8 1.0], ...
                        'EdgeColor','none',        ...
                        'FaceLighting','gouraud',  ...
                        'AmbientStrength', 0.15,   ...
                        'Parent', ax2);
       end
    end
```

```
    %Anwendung der Rotationsmatrix
    %         newangleview = double(campos - [mean_middlepoint 0]);
    %         view(ax2,newangleview);
    %% Festlegung der darauffolgenden Trackingpkt.
    % (ueberschreiben des Trackers)
    setPoints(pointtracker, currentvalidlocs);
    %% Visualisierung der Trackingpunkte aktualisieren
%     trackingmarkers = insertMarker( ....
%         currentframe, currentvalidlocs,'Size',7,'Color','yellow');
    figure(ar_plot);
%     imshow(trackingmarkers,'Parent',ax1,'Border','tight');
    middlepoint_marker = insertMarker( ...
        currentframe, mean_middlepoint, 'Size',7,'Color','red');
    imshow(middlepoint_marker,'Parent',ax1,'Border','tight');
    % iteration (Kameraufnahme-Schleife um einen erhoehen)
    counter = counter + 1;
    isValid = 0;
    M=getframe(ar_plot);
    if mod(counter,10)==0
    saveas(ar_plot,strcat(...
    'C:\Users\ricar\Matlab_Drive\Bachelorarbeit_AR_project\research\Mk2\run-
time_sequence\Frame',...
            num2str(counter),'.png'));
    end
end
%% ========================================================================
% reset: Schließen, Verbindung zur Hardware trennen und Speicher leeren
close all;
%% ========================================================================
% functions:
    function [mean_middlepoint, currentvalidlocs, ...
              currentframe, refimg, validpts, ...
              camFeatures, validrefpts,reffeatures] = ...
              initial_detection( ...
                firstrun, currentvalidlocs, pointtracker, cam, ax1)
        while firstrun == 1 || size(currentvalidlocs,1) < 30
            firstrun = 0;
            clear inlierCameraPts;
            [currentframe, refimg,...
            validpts, camFeatures, ...
            validrefpts,  reffeatures, ...
            matchedcampts] = ini_imgsource(cam,ax1);

            % Transformation zwischen den Referenzbilddaten
            % und den vorliegenden Bilddaten bestimmen
%             [refTransform, inlierindx, status] = ...
%                 estimateGeometricTransform2D(...
%                       matchedrefpts, matchedcampts,'projective');%'simi-
larity');

            release(pointtracker);
            if matchedcampts.Count >= 4
                initialize( ...
                    pointtracker, matchedcampts.Location, currentframe);
                [trackedpoints, isValid] = ...
                    step(pointtracker, currentframe);
                currentvalidlocs = trackedpoints(isValid,:);
                mean_middlepoint = mean(currentvalidlocs,1);
            end
            %disp("currentvalidlocs: "+num2str(size(currentvalidlocs,1)));
        end
```

```
        imshow(currentframe,'Parent', ax1,'Border','tight');
        % Darstellung der uebereinstimmenden Verbindungspunkte zwischen den
Bildern
        % (inlierpoints)
        % figure;
        % showMatchedFeatures(...
        %      currentframe, refimg, inlierCameraPts, inlierRefe-
rencePts,'Montage');
    end
    %% Initialisierung der Bilddaten;Erkennung von SURF Punkten
    % Extrahieren der features für die Übergabe an matchFeatures
    function [currentframe,refimg,...
              validpts,camFeatures, ...
              validrefpts,reffeatures, ...
              matchedcampts] = ini_imgsource(cam,ax1)
        % Initi. Referenzbild 'refimg' und Bildmerkmale erfassen:
            refimg = imread('firstimg.jpg');
            %refimg = imread('m4.jpg');
            % erkennen + extrahieren der 'SURF Features'
            refimggray = rgb2gray(refimg);
            refpts = detectSURFFeatures(refimggray);
            refpts = refpts.selectStrongest(75);
            [reffeatures, validrefpts] = ...
                extractFeatures(refimggray, refpts);
            % Darstellung der 'SURF Features'
%             figure;
%             imshow(refimg),hold on;
%             plot(refpts.selectStrongest(75));
            %figure;
        % init. kamera 'cam' und Bildmerkmale erfassen:
            % verbindung zur Kamera aufbauen
            % erkennen + extrahieren der 'SURF Features' in aktueller Auf-
nahme
            matchedcampts.Count = 0;
            while matchedcampts.Count < 4
                currentframe = snapshot(cam);
                currentframegray = rgb2gray(currentframe);
                campts = detectSURFFeatures(currentframegray);%,'NumOcta-
ves',4,'NumScaleLevels',6);
                campts = campts.selectStrongest(75);
                % darstellung der akt. Aufnahme
%                 hold off;
                imshow(currentframe,'Parent', ax1,'Border','tight');
%                 axis image;
%                 hold on;
                [camFeatures, validpts] = ...
                    extractFeatures(currentframegray, campts);
                %plot(validpts.selectStrongest(75));
                % Matching der Features zwischen Aufnahme und Referenzbild
                idxpairs = ...
                matchFeatures(camFeatures, reffeatures,'MaxRatio',0.9);%,...
                %'Method','Approximate','MatchThreshold',15);
                % speichern der jwl. matched surf pkt.
                matchedcampts = validpts(idxpairs(:,1));
                %=============== CHECK REUSE
                % matchedrefpts = validrefpts(idxpairs(:,2));
                %=============== CHECK REUSE
                %figure;
                %showMatchedFeatures(...
                %   currentframe, refimg, matchedcampts, matchedrefpts,'Mon-
tage');
                %disp("matchedcampts: "+num2str(matchedcampts.Count));
            end
```

```
%              hold off;
%              imshow(currentframe,'Parent', ax1,'Border','tight');
%              axis image;
    end
    %% STL Datensatz laden und in 'varargout' zur Uebergabe speichern
        %Eric Johnson (2021). STL File Reader
        %(https://www.mathworks.com/matlabcentral/fileexchange/22409-stl-
file-reader),
        %MATLAB Central File Exchange. Retrieved July 2, 2021.
        % stlread ist eine open source funktion:
        % Importiert stl-dateien nach Matlab...
        %{
        %...und gibt ein struktiertes Array aus
        %das von der Matlab Funktion "patch()"
        %interpretiert werden kann um
        %visualisiert zu werden.
        (Output/struct enthaelt Daten zu den Eckpunktkoordinaten 'Vertices'
        und Flaechen die durch die Eckpuntke aufgespannt werden 'Faces')
        %}
    function varargout = stlread(file)
    % STLREAD imports geometry from an STL file into MATLAB.
    %    FV = STLREAD(FILENAME) imports triangular faces from the ASCII or
binary
    %    STL file idicated by FILENAME, and returns the patch struct FV,
with fields
    %    'faces' and 'vertices'.
    %
    %    [F,V] = STLREAD(FILENAME) returns the faces F and vertices V sepa-
rately.
    %
    %    [F,V,N] = STLREAD(FILENAME) also returns the face normal vectors.
    %
    %    The faces and vertices are arranged in the format used by the PATCH
plot
    %    object.
    % Copyright 2011 The MathWorks, Inc.
        if ~exist(file,'file')
            error(['File ''%s'' not found. If the file is not on MATLAB''s
path' ...
                   ', be sure to specify the full path to the file.'],
file);
        end

        fid = fopen(file,'r');
        if ~isempty(ferror(fid))
            error(lasterror); %#ok
        end

        M = fread(fid,inf,'uint8=>uint8');
        fclose(fid);

        [f,v,n] = stlbinary(M);
        %if( isbinary(M) ) % This may not be a reliable test
        %    [f,v,n] = stlbinary(M);
        %else
        %    [f,v,n] = stlascii(M);
        %end

        varargout = cell(1,nargout);
        switch nargout
            case 2
                varargout{1} = f;
```

```
                varargout{2} = v;
            case 3
                varargout{1} = f;
                varargout{2} = v;
                varargout{3} = n;
            otherwise
                varargout{1} = struct('faces',f,'vertices',v);
        end
    end
    function [F,V,N] = stlbinary(M)
        F = [];
        V = [];
        N = [];

        if length(M) < 84
            error('MATLAB:stlread:incorrectFormat', ...
                  'Incomplete header information in binary STL file.');
        end

        % Bytes 81-84 are an unsigned 32-bit integer specifying the number
of faces
        % that follow.
        numFaces = typecast(M(81:84),'uint32');
        %numFaces = double(numFaces);
        if numFaces == 0
            warning('MATLAB:stlread:nodata','No data in STL file.');
            return
        end

        T = M(85:end);
        F = NaN(numFaces,3);
        V = NaN(3*numFaces,3);
        N = NaN(numFaces,3);

        numRead = 0;
        while numRead < numFaces
            % Each facet is 50 bytes
            %  - Three single precision values specifying the face normal
vector
            %  - Three single precision values specifying the first vertex
(XYZ)
            %  - Three single precision values specifying the second vertex
(XYZ)
            %  - Three single precision values specifying the third vertex
(XYZ)
            %  - Two unused bytes
            i1    = 50 * numRead + 1;
            i2    = i1 + 50 - 1;
            facet = T(i1:i2)';

            n  = typecast(facet(1:12),'single');
            v1 = typecast(facet(13:24),'single');
            v2 = typecast(facet(25:36),'single');
            v3 = typecast(facet(37:48),'single');

            n = double(n);
            v = double([v1; v2; v3]);

            % Figure out where to fit these new vertices, and the face, in
the
            % larger F and V collections.
```

```
            fInd  = numRead + 1;
            vInd1 = 3 * (fInd - 1) + 1;
            vInd2 = vInd1 + 3 - 1;

            V(vInd1:vInd2,:) = v;
            F(fInd,:)        = vInd1:vInd2;
            N(fInd,:)        = n;

            numRead = numRead + 1;
        end

    end
% ===============CHECK REUSE
%     function [F,V,N] = stlascii(M)
%         warning('MATLAB:stlread:ascii','ASCII STL files currently not sup-
ported.');
%         F = [];
%         V = [];
%         N = [];
%     end
%     % TODO: Change the testing criteria! Some binary STL files still begin
with
%     % 'solid'.
%     function tf = isbinary(A)
%     % ISBINARY uses the first line of an STL file to identify its format.
%         if isempty(A) || length(A) < 5
%             error('MATLAB:stlread:incorrectFormat', ...
%                   'File does not appear to be an ASCII or binary STL
file.');
%         end
%         if strcmpi('solid',char(A(1:5)'))
%             tf = false; % ASCII
%         else
%             tf = true;  % Binary
%         end
%     end
% ===============CHECK REUSE
%% ============================================================================
```

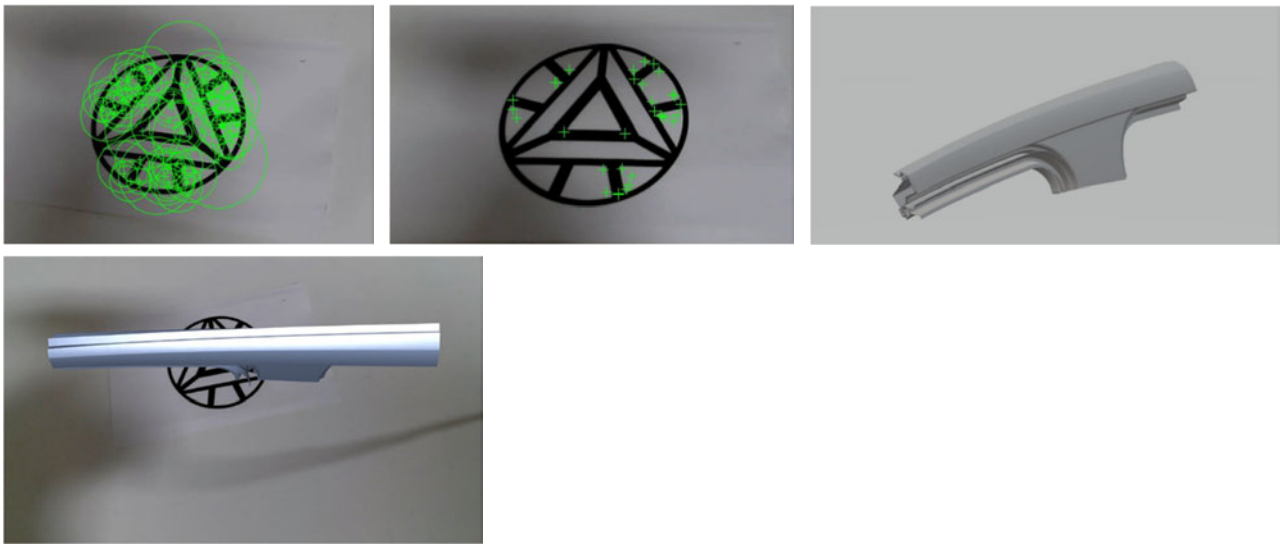
Anhang D: Kurzreferat

HAW Hamburg
Department Fahrzeugtechnik und Flugzeugbau
Berliner Tor 9

20099 Hamburg

Bachelorarbeit: **Erstellung einer Augmented Reality-Testumgebung mit Matlab**
Abgabedatum: **08.07.2021**
Verfasser: **Ricardo Möhler**

1. **Prüfer: Prof. Dr.-Ing. Adamski**
2. **Prüfer: Pro. Freytag**



Kurzreferat (Abstract):

Augmented Reality wird in der heutigen Produktentwicklung der Autoindustrie und als eigenständiger Produktbestandteil immer mehr durch Innovationen optimiert und für mehrere Bereiche zugänglicher.

Mit einem Einstieg in eine einfach umzusetzende Augmented Reality Testumgebung, mithilfe der Computer Vision Toolbox in Matlab, lässt sich eine erste Vorstellung über die Funktionsweise von AR und dem Ursprung von bedeutenden Algorithmen aus den Anfängen der erweiterten Realität und des computerbasierten Bildverarbeitung gewinnen. Dabei werden die bekanntesten Algorithmen vorgestellt und ein Überblick über die allgemeinen Bestandteile und Grundvoraussetzungen für das Erstellen einer eigenen AR Anwendung geschaffen. Ergänzend findet eine Testung, der in Matlab verfügbaren Funktionen, zur Entscheidungsfindung statt.

Am Beispiel eines virtuellen Objekts, in Form eines .jpg Bildes und einem .stl Datensatz, werden zwei Beispiele, für einen Prototypen, schrittweise entwickelt und eine Endversion davon zur Verfügung gestellt.

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: Möhler

Vorname: Ricardo

dass ich die vorliegende Bachelorarbeit ☐ bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Erstellung einer Augmented Reality Testumgebung

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -

Die Kennzeichnung der von mir erstellten und verantworteten Teile der -bitte auswählen- ist erfolgt durch:

Henstedt-Ulzburg	[redacted]	
Ort	Datum	Unterschrift im Original

Fakultät Technik und Informatik

HAW HAMBURG

Veröffentlichungshinweise

HAW HAMBURG

Thema:
Erstellung einer Augmented Reality-Testumgebung mit Matlab

Verfasser/in: Ricardo Möhler
Matrikelnummer: [redacted]
Studiengang: Fahrzeugbau BA

Erstprüfer/in: Prof. Dr. - Ing. Adamski
Zweitprüfer/in: Prof. Freytag

Dateiname auf der CDROM:
Dateigröße in MB :

Einwilligung zur elektronischen Veröffentlichung:
*Ich/Wir **stimme/n zu**, dass meine/unsere Abschlussarbeit durch die Hochschule für Angewandte Wissenschaften Hamburg im Internet veröffentlicht wird. Meine/Unsere Urheberrechte als Autor bleiben von dieser Einwilligung unberührt.*

Hamburg, den [redacted] _____
Unterschrift/en

Fakultät Technik und Informatik
Fakultätsservicebüro

Faculty of Engineering and Computer Science
Faculty Service Office

1. Füllen Sie das Formular am Computer aus.
2. Drucken Sie das Formular 3 Mal aus
3. Unterschreiben Sie alle 3 Formulare
4. Schneiden Sie bei einem Exemplar den Rahmen aus und benutzen dies als Cover für Ihre CD (Abschlussarbeit im PDF Format)
5. Geben Sie die anderen 2 Exemplare zusammen mit Ihrer Abschlussarbeit ab
6. Die Unterschriften der Prüfer müssen nicht von Ihnen eingeholt werden

Bitte beschriften Sie auch Ihre CDROM entsprechend!

▼ Zur elektronischen Veröffentlichung geeignet: ☐ ja ☐ nein

_____ __ __ . __ __ . 20 __ __
Erstprüfer/in Datum

_____ __ __ . __ __ . 20 __ __
Zweitprüfer/in Datum

▼ Weiterleitung des Datenträgers mit Begleitblatt an die Bibliothek:

Datum __ __ . __ __ . 20 __ __ _____
Unterschrift Fakultätsservicebüro

▼ OPUS_ID: _____

Bitte zwei Kopien dieser Veröffentlichungshinweise mit der Abschlussarbeit einreichen!

Fakultät Technik und Informatik

HAW HAMBURG

Veröffentlichungshinweise

HAW HAMBURG

Thema:

Erstellung einer Augmented Reality-Testumgebung mit Matlab

Verfasser/in: Ricardo Möhler

Matrikelnummer: [redacted]

Studiengang: Fahrzeugbau BA

Erstprüfer/in: Prof. Dr. - Ing. Adamski

Zweitprüfer/in: Prof. Freytag

Dateiname auf der CDROM:

Dateigröße in MB:

Verweigerung zur elektronischen Veröffentlichung:
*Ich/Wir **verweigern** die Veröffentlichung meine/unsere Abschlussarbeit durch die Hochschule für Angewandte Wissenschaften Hamburg im Internet.*

Hamburg, den [redacted] _____

Unterschrift/en

Fakultät Technik und Informatik
Fakultätsservicebüro

Faculty of Engineering and Computer Science
Faculty Service Office

1. Füllen Sie das Formular am Computer aus.
2. Drucken Sie das Formular 3 Mal aus
3. Unterschreiben Sie alle 3 Formulare
4. Schneiden Sie bei einem Exemplar den Rahmen aus und benutzen dies als Cover für Ihre CD (Abschlussarbeit im PDF Format)
5. Geben Sie die anderen 2 Exemplare zusammen mit Ihrer Abschlussarbeit ab
6. Die Unterschriften der Prüfer müssen nicht von Ihnen eingeholt werden

Bitte beschriften Sie auch Ihre CDROM entsprechend!

Die Abschlussarbeit mit dem oben angegebenen Titel darf **nicht** durch die Hochschule für Angewandte Wissenschaften veröffentlicht werden.

Begründung der Verweigerung:

_____

_____

_____

▼ Kenntnis von der Verweigerung der Veröffentlichung der oben angegebenen Arbeit

_____ __ __ . __ __ . 20 __ __
Erstprüfer/in Datum

_____ __ __ . __ __ . 20 __ __
Zweitprüfer/in Datum

Bitte zwei Kopien dieser Veröffentlichungshinweise mit der Abschlussarbeit einreichen!