UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Fraunhofer IDM@NTU
Fraunhofer IDM@NTU Singapur

Automatic Initialization of Model-Based 3D Tracking

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von
Markus Solbach

Erstgutachter: Prof. Dr. Stefan Müller
Institut für Computervisualistik, Koblenz

Zweitgutachter: M.Sc. Martin Buchner
Fraunhofer IDM@NTU, Singapur

Koblenz, im Juni 2013

!!!Ersetzen durch Aufgabenstellung!!!

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....
(Ort, Datum) (Unterschrift)

Abstract

Augmented Reality (AR) is getting more and more popular. To augment information into the field of vision of the user using HMDs, e.g. front shields of a car, glasses, displays of a smartphone or tablets are the main use of AR technology. It is necessary to get the position and orientation (pose) of the camera in space to augment correctly.
Nowadays, this is solved with artificial markers. These known markers are placed in the room and the system is taught to this set up. The next step is to get rid of of these artificial markers. If we are calculating the pose without such markers we are talking about marker-less tracking. Instead of artificial markers we will use natural objects in the real world as reference points to calculate the pose. Thus, this approach can be used flexible and dynamically. We are no longer dependent on artificial markers but we need much more knowledge about the scenery to find the pose. This is compensated by technical actions and/or the user himself. However, both solutions are neither comfortable nor efficient for the usage of such a system. This is why marker-less 3D tracking is still a big field of research.
This sets the starting point for the bachelor thesis. In this thesis an approach is proposed that needs only a quantity of 2D Feature from a given camera image and a quantity of 3D Feature of an object to find the initial Pose. With this approach, we got rid of the technical and user assistance. 2D and 3D Features can be detected in any way you like.
The main idea of this approach is to build six correspondences between this quantities. With those we are able to estimate the pose. Each 3D Feature is mapped with the estimated pose onto image coordinates, whereby the estimated pose can be evaluated. Each distance is measured between the mapped 3D Feature and the associated 2D Feature. Each correspondency is evaluated and the results are summed up to evaluate the whole pose. The lower this summed up value is, the better the pose. It has been shown to have a correct pose with a value around ten pixels.
Due to lots of possibilities to build six correspondences between the quantities, it is necessary to optimize the building process. For the optimization we will use a genetic algorithm.
During the test case the system worked quite reliable. The hit rate was around 90% with a runtime of approximately twelve minutes. Without optimization it can take easily some years.

Abstract

Augmented Reality erfreut sich wachsender Beliebtheit. Zusatzinformationen in HMDs, Windschutzscheiben oder im Kamerabild des Smartphones oder Tablets sind hier die nennenswertesten Anwendungsfälle. Um eine Einblendung korrekt anzuzeigen, ist es notwendig die Position und Orientierung (Pose) der Kamera im Raum zu erfassen. Dies geschieht zurzeit hauptsächlich Unterzuhilfenahme von Markern. Dabei werden vordefinierte Marker im Raum positioniert und das System angelernt, wie es diese zu interpretieren hat. Der nächste Schritt ist es ohne Marker auszukommen. Hierbei wird von dem markerlosen Tracking gesprochen. Anstelle von künstlichen Markern werden natürliche Objekte der realen Umgebung als Referenzpunkte genutzt, um die Kamerapose zu bestimmen. Dadurch lässt sich dieses Verfahren flexibel und dynamisch einsetzen. Es wird zwar auf die Zuhilfenahme von Markern verzichtet, aber ein größeres Vorwissen über die Szenerie ist notwendig. Dies wird über technische Maßnahmen realisiert und/oder durch Interaktion des Benutzers. Beides ist nicht komfortabel oder effizient in der Verwendung eines solchen Systems und ist ein Grund dafür, warum markerloses 3D-Tracking nach wie vor ein Forschungsbereich ist.
An diesem Punkt setzt diese Arbeit an. Es wird ein Ansatz vorgeschlagen, der lediglich eine Menge von 2D-Feature und eine Menge von 3D-Feature eines Objekts benötigt, um die initiale Pose zu finden. Es sind keine weiteren technischen Hilfen notwendig und auch auf die Interaktion mit dem Benutzer wird verzichtet. Die 2D-Feature, wie auch die 3D-Feature, können auf beliebige Art gewonnen werden.
Die Idee ist es, diese zwei Mengen mit sechs Korrespondenzen zu verbinden. Anhand dieser Korrespondenzen kann eine Pose geschätzt werden. Mit der erhaltenen Pose kann jedes 3D-Feature auf Bildkoordinaten abgebildet werden, wodurch sich die geschätzte Pose bewerten lässt. Dabei wird der Abstand zwischen abgebildetem 3D-Feature und seinem zugehörigen 2D-Feature gemessen. Jede Korrespondenz wird so bewertet und die Ergebnisse aufsummiert. Je niedriger die Summe, desto besser ist die Pose. Es hat sich gezeigt, dass ein Wert von zehn Pixeln bereits ausreichend ist, um eine Pose als richtig zu bewerten.
Da es sehr viele Möglichkeiten gibt, diese sechs Korrespondenzen zwischen beiden Mengen aufzubauen, muss dieses Verfahren optimiert werden. Dies geschieht mit einem genetischen Algorithmus.
In dem Testszenario arbeitet das fertige System sehr zuverlässig. Es wird eine Trefferquote von ca. 90%, bei einer Laufzeit von ungefähr zwölf Minuten, erreicht. Ohne Optimierung kann das Finden der initialen Pose schnell mehrere Jahre dauern.

Danksagung

Ich möchte mich an dieser Stelle bei all denen bedanken, die mich bei meinem sechsmonatigen Aufenthalt in Singapur zur Erstellung dieser Arbeit unterstützt haben.
Ein ganz besonderer Dank ist an Professor Müller zu richten. Ohne Sie wäre dieser Auslandsaufenthalt erst gar nicht möglich gewesen. Zudem ein großes Dankeschön für die gute Betreuung und wertvollen Tipps in unseren regelmäßigen Ferngesprächen. Selbstverständlich gilt dieser Dank auch Professor Müller-Wittig und dem gesamten Team des Instituts. Ich wurde hervorragend unterstützt, herzlich aufgenommen und konnte in einer für mich perfekten Umgebung diese Bachelorarbeit anfertigen. Hierbei ist Martin Buchner besonders hervorzuheben. Du hattest für meine Fragen und teilweise verrückten Ideen immer ein offenes Ohr und hast mir mit deinen Ratschlägen und Denkanstößen sehr geholfen. Ich kann mit Sicherheit sagen, dass ich keinen besseren Betreuer hätte finden können. Des Weiteren möchte ich mich bei Marcus Poicke und seiner Frau Ora bedanken. Ihr wart maßgeblich dafür verantwortlich, dass mein Aufenthalt so toll war. Ihr habt es mir sehr schwer gemacht Singapur wieder zu verlassen.
Zu guter Letzt möchte ich aus tiefstem Herzen denen danken, denen ich nicht genug danken kann. Meinen Eltern.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die erweiterte Realität[1] ist seit Mitte 1990er Jahre ein immer weiter wachsendes Forschungsfeld der Informatik. Vor allem in den Disziplinen der Computergrafik und Computer Vision.

Es bezeichnet die Einbindung einer virtuellen Realität in die reale. Wobei die virtuelle Realität aus einem einfachen Schriftzug, Annotationen, Bildern, Videos oder auch komplexen 3D-Objekten bestehen kann. In den meisten Anwendungsfällen wird versucht, eine möglichst hohe Immersion[2] zu erzeugen. Der Benutzer soll die virtuellen Einblendungen nicht als störend empfinden. Viel mehr als real oder natürlich.
Als mögliches Anwendungsbeispiel seien hier sogenannte HMDs (Head Mounted Display) genannt, die sich vor allem in den letzten Jahren wachsender Beliebheit erfreuen und mit der Google® Glass nun auch der breiten Masse zur Verfügung stehen. Direkt eingeblendete Informationen in die Brille, beispielsweise zur Navigation, sind nicht mehr ferne Zukunftsmusik aus Hollywood. Informationen zu erhalten, aufzunehmen und auszutauschen wird immer mehr in den Alltag integriert werden.
Diese Technologie basiert auf einer nicht trivialen Information der Pose. Sie beschreibt die Position und Orientierung eines Objekts im Raum[1]. In der erweiterten Realität ist es das Objekt, über das man Informationen erhalten möchte oder der Benutzer selbst. Für Navigationsanwendungen im öffentlichen Verkehr reichen approximierte Posen, die auf mehrere Meter genau sind. Diese können durch Gleichfeldmessungen anhand eines Kompass (Orientierung) und Messungen der Time of Flight[3], wie im GPS (Global Positioning System) (Position) errechnet werden. Bei solchen Anwendungsbeispielen spricht man von einem Grobtracking.
Tracking[4] wird neben dem Grobtracking in das Feintracking unterteilt. Es wird häufig dazu verwendet, eine Pose zu verfeinern, die das Ergebnis

[1] auch englisch *augmented reality*, kurz *AR*
[2] spätlat. immersio = Eintauchung
[3] Messung der Übertragungszeit eines Signals
[4] Tracking = erfassen der Pose

eines Grobtrackings ist. Hierzu haben sich optische Verfahren durchgesetzt. Sie werden in zwei Gruppen unterteilt. Zum einen in eine sogenannte Inside-out Konfiguration. Bei ihr ist das zu erfassende Objekt die Kamera selbst und es wird über Referenzen im Raum die Pose ermittelt. Zum anderen in eine Outside-in Konfiguration. Sie beschreibt das Ermitteln der Pose eines Objekts im Raum. Hierbei dient die Kamera als Referenz.
Als Referenzen unterscheidet man aktive und passive Marker. Aktive Marker sind selbstleuchtend. Sie erzeugen markante Stellen im aufgenommenen Bild, meistens helle Punkte und können ausgewertet werden. Als vorteilhaft hat sich hierbei die Kombination aus Markern, die im für Menschen nicht sichtbaren Infrarotbereich leuchten, und einer Infrarotkamera erwiesen. Der Benutzer wird von den Markern nicht gestört und dadurch, dass die Infrarotkamera nur das Licht in einem bestimmten Spektrum wahrnimmt, ist auch die Auswertung stabiler. Das Gegenstück stellen passive Marker dar. Sie besitzen eine eindeutige Geometrie, die in dem aufgenommen Kamerabild zu finden ist.
Der Nachteil von markerbasiertem Tracking ist ihr technischer Aufwand. Möchte man eine hohe Immersion gewährleisten, sollte man darauf achten, dass der Benutzer die Marker nicht wahrnimmt. Dies ist mit Markern allgemein nur sehr schwer umsetzbar. Man könnte versuchen, die Marker so zu integrieren, dass sie eine Einheit mit der Umwelt ergeben, was aber direkt ein weniger stabiles optisches Tracking bedeutet. Ein optisches Tracking ist umso stabiler, desto markantere Merkmale, im weiteren Verlauf der Arbeit Feature genannt, vorhanden sind. Passt man dieses Merkmal nun an, verschwimmt es im restlichen Bild und kann nicht mehr eindeutig identifiziert werden. In der Regel, wie in [2] angegeben, müssen passive Marker sogar ausgeleuchtet werden, damit sie sich von ihrer Umgebung abheben und besser vom verarbeitenden System erkannt werden können.

Aus den zuvor genannten Gründen hat sich ein Forschungsbereich aufgetan, der sich damit befasst auf Marker zu verzichten: Das sogenannten markerlosem Tracking. Bis zu der Erstellung dieser Arbeit war das Problem der initialen Posenfindung mit nur wenig Vorwissen nicht gelöst. Wäre dieses Problem gelöst, würde es einen enormen Schritt für die Bereiche der erweiterten Realität, Robotik, Medizin, Maintenance, industrielle Fertigung etc. bedeuten.
Diese Arbeit befasst sich mit einem selbst entwickelten Ansatz des markerlosen Trackings, der zur Berechnung der initialen Pose lediglich mit einem 3D-Feature Modell des Referenzobjekts und extrahierten 2D-Feature aus einem Kamerabild auskommt. Die Idee ist, zwischen den 3D und 2D-Feature Korrespondenzen herzustellen und aus Ihnen eine Pose zu errechnen. Mit ihr wird jedes 3D-Feature einer Korrespondenz mathematisch auf 2D-Bildkoordinaten abgebildet und die Distanz zu seinem 2D-Feature gemessen. Liegt eine geringe Distanz aller Korrespondenzen vor, wurde die

Pose gefunden.

1.2 Zielsetzung

Diese Bachelorarbeit versucht die Frage zu beantworten, wie die initiale Pose unter alleiniger Nutzung eines 3D-Feature-Modell gefunden werden kann. Alle bisher vorgestellten Verfahren der initialen Posenfindung basieren auf der Verwendung von weiteren Sensoren wie GPS und Kompass und/oder einer Mensch-Computer-Interaktion. Beide Gruppen haben große Nachteile. Die erste Gruppe funktioniert beispielsweise nicht in geschlossenen Räumen, da sie GPS nutzt. Die zweite Gruppe setzt ein räumliches Vorstellungsvermögen des Benutzers voraus. Viele Menschen haben Schwierigkeiten bezogen auf das räumliche Vorstellungsvermögen oder besitzen es nicht. Das führt dazu, dass viele Nutzer der Software mit Ablehnung gegenübertreten.
Wie in Kapitel 4.2 gezeigt wird, ist die initiale Posenbestimmung nicht trivial. Dies hauptsächlich in der Größe des Suchraums begründet liegt. Um das Problem der Größe zu lösen, ist es notwendig das Durchlaufen des Suchraums mit einem *genetischen Algorithmus* zu optimieren.

1.3 Aufbau der Arbeit

Die Arbeit ist in acht Kapitel aufgeteilt. In Kapitel 2 (Grundlagen) sollen die Grundlagen, die zum Lösen der Problemstellung nötig sind, erläutert werden. Dabei wird auf Verfahren zur 2D-Feature und 3D-Feature Gewinnung, Verfahren zur Posenschätzung, Optimierungsverfahren und die mathematische Abbildung von 3D-Feature in Bildkoordinaten eingegangen. Kapitel 3 (Stand der Technik) befasst sich mit dem Stand der Technik. Da es bis zum Zeitpunkt dieser Arbeit keinen Lösungsvorschlag zum Finden der initialen Pose unter alleiniger Nutzung eines 3D-Modells gab, werden Arbeiten angeführt, die mit diesem Problem ebenfalls konfrontiert waren. Dabei wird sich zeigen, dass die initiale Pose unter Verwendung von weiteren Sensoren wie GPS und Kompass und/oder einer Mensch-Computer-Interaktion gefunden wird. Im darauffolgenden Kapitel 4 (Konzept) wird das Konzept vorgestellt. Dabei wird jeder Baustein, der die Idee dieser Arbeit ausmacht, genau beschrieben. Daran schließt sich Kapitel 5 (Realsierung) an. Es gibt Hinweise, wie das zuvor beschriebene Konzept implementiert wurde. Im vorletzten Kapitel 6 (Evaluation) wird der Lösungsansatz durch verschiedene Versuche untersucht. Da die Herangehensweise mit einem solchen Optimierungsverfahren neu ist, wird ein Hauptaugenmerk auf die Auswirkung der Parameter des *genetischen Algorithmus* gelegt. Zum Schluss wird in Kapitel 7 (Zusammenfassung) ein Resümee der vorliegenden Bachelorarbeit gegeben und mit einem Ausblick abgeschlossen.

2 Grundlagen

Im folgendem Kapitel sollen die Grundlagen erläutert werden, die für das Lösen der Problemstellung nötig sind.

2.1 Feature

Unter einem Feature im Kontext der Bildverarbeitung versteht man ein markantes Merkmal im Bild und dies wird auch als Bildmerkmal bezeichnet.
Features bezeichnen lokale Bildmuster, die sich in ihrer direkten Nachbarschaft unterscheiden und in mindestens einer Bildeigenschaft eine Änderung vorzeigen. Als Beispiele kann man hier Kontraständerungen, Kanten, Ecken, geometrische Formen und Farbsegmente nennen. Weiter unterscheiden kann man Feature in 2D und 3D-Feature. Dabei gibt der Dimensionenbezeichner lediglich die Art des Eingabeformats an.
Diese Arbeit nutzt zur 2D-Feature Gewinnung Punktdetektoren. Die Qualität eines Punktdetektors kann nach [3] mit vier Kriterien bewertet werden:

- *Detektionsrate:* Beschreibt das Maß für die Anzahl erfolgreicher Detektionen
- *Wiederholbarkeitsrate:* Gibt die Stabilität der Detektionen unter verschiedenen Bedingungen (Transformation, Helligkeit, Rauschen) an
- *Lokalisationsgenauigkeit:* Ist ein Maß für den durchschnittlichen Pixelabstand zwischen gefundener Ecke und tatsächlicher Ecke im Bild
- *Rechenaufwand:* Beschreibt die Laufzeit des Algorithmus, um Ecken im Bild zu detektieren

Die wichtigsten Verfahren, um ein grundlegendes Verständnis der 2D- Feature Extraktion zu erhalten, sollen nun kurz vorgestellt werden.

2.1.1 Movarec Punktdetektor

Hans Moravec veröffentlichte 1977 den nach ihm benannten Operator [4]. Er zählt zu den ersten Punktdetektoren, die es gab. Das Verfahren definiert eine Ecke auf dem Pixel $I_{x,y}$ des Eingabebildes I, an dem die resultierende

Intensitätsverschiebung $V_{x,y}$ über einem Schwellwert liegt.
$V_{x,y}$ beschreibt das Minimum der Summe der quadratischen Differenz zwischen benachbarten Pixeln in einer der vier Hauptrichtungen $V^1_{x,y}$, $V^2_{x,y}$, $V^3_{x,y}$, $V^4_{x,y}$ des Fensters der Größe $m \times n$.

$$V_{x,y} = min(V^1_{x,y}, V^2_{x,y}, V^3_{x,y}, V^4_{x,y}) \tag{2.1.1}$$

$$V^1_{x,y} = \sum_{x=-m}^{m} \sum_{y=-n}^{n-1} (I_{x,y} - I_{x,y+1})^2 \tag{2.1.2}$$

$$V^2_{x,y} = \sum_{x=-m}^{m-1} \sum_{y=-n}^{n} (I_{x,y} - I_{x+1,y})^2 \tag{2.1.3}$$

$$V^3_{x,y} = \sum_{x=-m}^{m-1} \sum_{y=-n}^{n-1} (I_{x,y} - I_{x+1,y+1})^2 \tag{2.1.4}$$

$$V^4_{x,y} = \sum_{x=-m}^{m-1} \sum_{y=-n}^{n-1} (I_{x,y+1} - I_{x+1,y})^2 \tag{2.1.5}$$

Der *Movarec Operator* dient als Grundlage für viele später entstanden Detektoren (Harris (2.1.2), Förstner, Kanade-Lucas-Tomasi). Er besitzt, wie in [3] angegeben, eine geringe Wiederholbarkeitsrate sowie eine hohe Rechenzeit. Das führt dazu, dass er nur noch selten zur Merkmalsextraktion benutzt wird. Vorteilhaft ist zu nennen, dass er leicht zu implementieren ist.

2.1.2 Harris Punktdetektor

Als Verbesserung des *Movarec Punktdetektor* (2.1.1) wurde 1988 von Chris Harris und Mike Stephens der *Harris Punktdetektor* [5] entwickelt (auch *Plessy Punktdetektor* genannt).
Sie ersetzten die diskrete Verschiebung der vier Hauptrichtungen $V^1_{x,y}$, $V^2_{x,y}$, $V^3_{x,y}$, $V^4_{x,y}$ mit Hilfe der Autokorrelationsfunktion A. Damit wurde auch die Lokalisationsgenauigkeit erhöht.
Die Kreuzkorrelation eines Signals mit sich selbst wird als Autokorrelationsfunktion bezeichnet. Im Kontext der Bildverarbeitung sind Signale als die Ableitung eines Bildes in i- und j-Richtung zu verstehen.

In [5] werden drei Probleme des *Movarec Punktdetektor*(2.1.1) genannt und Lösungen vorgeschlagen:

- *Diskretisierung auf 45 Grad:* Aufgrund der Intensitätsverschiebung $V_{x,y}$ (2.1.1) in nur vier Richtungen ist das Ansprechverhalten des Operators anisotrop.

  - Alle möglichen Verschiebungen können mit einer analytischen Erweiterung des Verschiebungsursprungs abgedeckt werden. (s. [5])

- *Verrauschtes Ergebnis:* Da das Suchfenster binär und rechtwinklig ist.
  - Es wird vorgeschlagen ein kreisförmiges Weichzeichnungsfilter, wie einem Gaußfilter, zu verwenden.

- *Detektierung zu vieler Ecken:* Es wird lediglich das Minimum der Intensitätsverschiebung $V_{x,y}$(2.1.1) zur Eckendetektion in Betracht gezogen.
  - Es sollen alle Intensitätsverschiebung $V_{x,y}$(2.1.1) in Betracht gezogen werden.

Zunächst soll angegeben werden, wie sich eine Autokorrelationsmatrix in einem diskreten Bild aufbauen lässt. Definitionen und Symbole wurden dazu in leicht abgeänderter Form aus [6] entnommen.

Seien zunächst

$$I_x := I * K^{(x)}, \text{ das Bild } I \text{ abgeleitet in } x\text{-Richtung.} \tag{2.1.6}$$

$$I_y := I * K^{(y)}, \text{ das Bild } I \text{ abgeleitet in } y\text{-Richtung.} \tag{2.1.7}$$

Wobei $K^{(x)}$ und $K^{(y)}$ einen Ableitungskern in x-, beziehungsweise y- Richtung bezeichnen. Für einen Bildpunkt $p \in [0, x_{max}[\times [0, y_{max}[$ kann nun ein Ableitungsvektor $\overrightarrow{p}'$ angegeben werden:

$$\overrightarrow{p}' = \begin{pmatrix} I_x(p) \\ I_y(p) \end{pmatrix} \tag{2.1.8}$$

Daraus folgt

$$A = \overrightarrow{p}'\overrightarrow{p}'^T = \begin{bmatrix} I_x^2(p) & I_x(p) \cdot I_y(p) \\ I_x(p) \cdot I_y(p) & I_y^2(p) \end{bmatrix} \tag{2.1.9}$$

die zu dem Punkt p gehörende Autokorrelationsmatrix. In [5] wird nicht nur ein Punkt zur Eckendetektion genutzt, sondern eine Punktregion. Die Region sei mit $(m, n) \in \Omega$ definiert. Daraus ergibt sich folgende Autokorrleationsmatrix A für eine Punktregion:

$$A_{m,n} = \begin{bmatrix} \sum\limits_{(m,n)} I_x^2(p), mn & \sum\limits_{(m,n)} I_x(p), mn \cdot I_y(p), mn \\ \sum\limits_{(m,n)} I_x(p), mn \cdot I_y(p), mn & \sum\limits_{(m,n)} I_y^2(p), mn \end{bmatrix} \tag{2.1.10}$$

Der Harris-Operator in der Region (m, n) ist definiert als

$$H_{m,n} = \det(A_{m,n}) - \kappa \cdot spur(A_{m,n})^2 \tag{2.1.11}$$

Wobei κ eine empirische Konstante zwischen 0,04 und 0,06 [7] ist. Aus Formel 2.1.11 lässt sich eine Interessenskarte des Eingabebilds I bilden. Diese wird mit einem zuvor festgelegtem Schwellwert ausgewertet und nach lokalen Maxima untersucht.
Ein großer Nachteil des Detektors ist seine Laufzeit [3]. Sie ist umfangreicher als die des *Movarec Punktdetektor* (2.1.1), was mit der aufwendigen Berechnung von $H_{m,n}$(2.1.11) begründet werden kann.

2.1.3 FAST Punktdetektor

Zuletzt soll der *FAST Punktdetektor* (Features from Accelerated Segment Test) [8] kurz vorgestellt werden. Er wurde 2006 von Edward Rosten und Tom Drummong vorgestellt und wird heute häufig zur Merkmalsextraktion genutzt.
Zurückzuführen ist der Detektor auf die Klasse der *AST* (Accelerated Segment Test) basierten Feature Detektoren. Dieser Test ist eine Vereinfachung des *SUSAN* (Smallest Univalue Segment Assimilating Nucleus) [9]. Der *SUSAN* Algorithmus verwendet den kompletten Kreis (in 1 schwarz und grün gekennzeichnet), der mit seinem Mittelpunkt auf den zu betrachtenden Pixel p (in 1 rot gekennzeichnet) gelegt wird. *AST* verwendet hingegen nur die Pixel, die von einem Bresenham Kreis (in 1 schwarz gekennzeichnet) mit einem Radius τ um p beschrieben werden. Der Bresenham-Algorithmus erstellt einen Kreis, dessen Besonderheit es ist, Rundungsfehler, die durch die Diskretisierung von kontinuierlichen Koordinaten entstehen, zu minimieren. Im Grunde können auch andere Algorithmen genutzt werden. Dieser wird jedoch vom Autor empfohlen. Als weiterführende Quelle sei [10] angegeben.
Sind entweder alle n zusammenhängende Pixel heller oder dunkler als p, dann liegt unter Berücksichtigung des Schwellwerts t ein Feature vor. Obwohl τ beliebig gewählt werden kann, benutzt *FAST* ausschließlich einen Radius von drei Pixel um p. Daraus ergibt sich ein Kreisumfang sowie betrachtete Nachbarn von 16. n wird in der Regel mit 12 angegeben, sofern nicht die in [11, 8] vorgeschlagene Verbesserung durch machine learning (2.1.3.2) genutzt wird.

2.1.3.1 High Speed Test

Eine erhebliche Performanzsteigerung kann durch den High Speed Test erreicht werden. Zu Beginn werden 4 Pixel (1, 9, 5 und 13) in gleichmäßigen Abständen auf dem Kreis gewählt (in Abbildung 2 blau markiert). Da laut Autor des Algorithmus 12 Pixel entweder heller oder dunkler als p sein

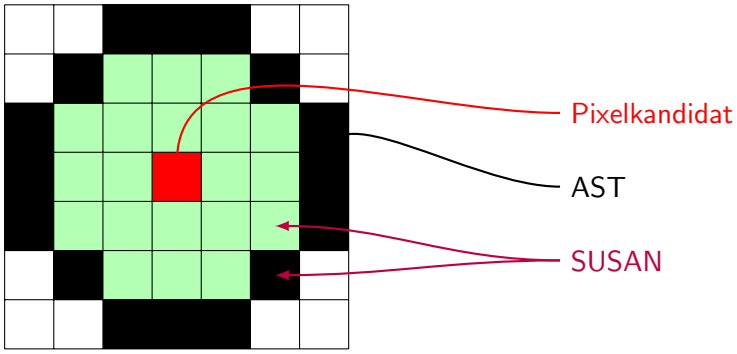

Abbildung 1: Unterschied SUSAN und AST
$p = 3$. Quelle: eigene Erstellung

müssen, müssen mindestens 3 Pixel aus diesen 4 heller oder dunkler als p sein.

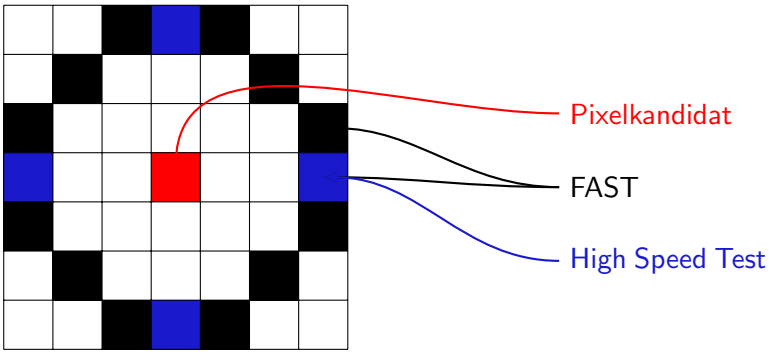

Abbildung 2: Unterschied FAST und FAST mit High Speed Test. Quelle: eigene Erstellung

Als erstes werden Pixel 1 und 9 getestet. Wenn beide Pixel-Intensitäten $val(p_1)$ und $val(p_9)$ innerhalb von $[val(p_p) - t, val(p_p) + t]$ liegen, kann p als Pixelkandidat verworfen werden. Ansonsten wird der Test mit Pixel 5 und 13 durchgeführt. Sind 3 Pixel aus diesem Bereich entweder heller als $val(p_p) + t$ oder dunkler als $val(p_p) - t$, werden die restlichen Pixel ausgewertet.

Nach [12] ermöglicht dieser Test, dass im Durchschnitt 3,8 Pixel benötigt werden, um p als Pixelkandidat zu identifizieren. Verglichen mit 16 Pixeln ist 3,8 eine enorme Leistungssteigerung. Diese Testmethode weist allerdings drei Schwachstellen auf:

- Der High-Speed-Test kann nicht generalisiert werden für $n < 12$

- Die Effizienz des Detektors hängt stark von der Wahl der Testpixel (1, 5, 9 und 13) ab.
- Es werden zu viele Pixel in direkter Nachbarschaft gefunden.

2.1.3.2 Verbesserung durch maschinelles Lernen

Die ersten beiden Punkte aus Kapitel 2.1.3.1 versucht der Autor durch einen *machine learning Ansatz* zu lösen. Er arbeitet in zwei Schritten. Als Erstes wird der Algorithmus auf Trainingsbilder mit einem definiertem n ausgeführt, wobei die Wahl der Trainingsbilder vorzugsweise auf die spätere Anwendungsdomäne abgestimmt sein sollte. Obwohl diese Arbeit ohne Trainingsbilder arbeitet und somit ohne Verbesserung durch *machine learning*, wird zur Vollständigkeit dieser Schritt des *FAST* Algorithmus kurz beschrieben.
Zu Beginn werden die Parameter des Algorithmus auf $\tau = 3$, also 16 Pixel, und t auf einen angemessenen Wert gesetzt.

Einem Pixelkandidat p kann für jeden Pixelort $loc \in \{1, 2, 3, ..., 16\}$ mit $S_{p \to loc}$ einer der drei Zustände (d, s, b) zugewiesen werden:

- d, $I_{p \to loc} \leq I_p - t$, für dunkler
- s, $I_p + t \leq I_{p \to loc} \leq I_p - t$, für ähnlich
- b, $I_{p \to loc} \geq I_p + t$, für heller

Beim Wählen eines loc kann P in drei verschiedene Untermengen P_d, P_s, P_b eingeteilt werden, wobei P die Menge aller Pixel in allen Trainingsbildern darstellt.

- $P_d = \{\, p \in P : S_{p \to loc} = d \,\}$
- $P_s = \{\, p \in P : S_{p \to loc} = s \,\}$
- $P_b = \{\, p \in P : S_{p \to loc} = b \,\}$

Zweitens wird ein Entscheidungsbaum aufgebaut, um aus den 16 Pixelorten loc die größte Kullback-Leibler-Divergenz zu erhalten. Die Kullback-Leibler-Divergenz bezeichnet ein Maß für die Unterscheidbarkeit zweier Wahrscheinlichkeitsverteilungen. Für den Entscheidungsbaum wird der Algorithmus ID3 (Iterative Dichotomiser 3) [13] genutzt. K_p eine Booleanvariable, die angibt, ob p eine Ecke ist. Die Entropie von K_p wird dazu genutzt, um die Information, ob es sich bei p um eine Ecke handelt, auszuwerten.
Die nicht normalisierte Entropie K_Q für eine Menge Pixel Q ergibt sich als:

$$H(Q) = (c + n) \log_2(c + n) - c \log_2 c - n \log_2 n, \tag{2.1.12}$$

wobei c die Anzahl der Ecken und n die Anzahl der Pixel ist, die keine Ecken sind. Die Kullback-Leibler-Divergenz kann nun als

$$H(g) = H(P) - H(P_b) - H(P_s) - H(P_d) \tag{2.1.13}$$

angegeben werden. Rekursiv wird ein Lösung gesucht, die Formel 2.1.13 maximiert. Das Abbruchkriterium der Rekursion ist erfüllt, wenn die Entropie 0 ist, also entweder alle Pixel in einer Untermenge Ecken sind oder keine.
Übersetzt in eine Programmiersprache wie C oder C++ besteht der Entscheidungsbaum hauptsächlich aus verschachtelten if- und else- Anweisungen (3) und kann direkt zur Eckendetektion genutzt werden.

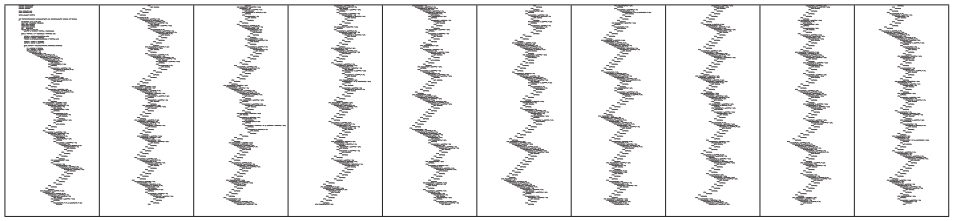

Abbildung 3: Etwa ein Drittel des in C übersetzten Entscheidungsbaumes. (Nur zur Veranschaulichung) Quelle: [11]

2.1.3.3 Nicht-Maximum Unterdrückung

Eine Nicht-Maximum Unterdrückung beschreibt den letzten Schritt des *FAST*-Algorithmus. Da bisher dem Pixelkandidt p kein Wert zugewiesen werden konnte, an dem man seine Eckigkeit messen kann, kann eine Nicht-Maximum Unterdrückung nicht direkt durchgeführt werden.
Die Eckigkeit wird definiert als der größte Wert für t, der p noch als eine Ecke detektiert. Zwei Ansätze werden dazu vorgeschlagen:

- Anhand einer binären Suche kann nach dem größten t gesucht werden, das p als eine Ecke detektiert
- Der zweite Vorschlag ist ein iterativer Ansatz. Hierbei wird t verkleinert, bis p nicht mehr als Ecke detektiert wird.

2.2 3D-Feature-Modell

Im Kontext dieser Arbeit ist ein 3D-Feature-Modell eine Menge an 3D-Punkten, die ein Objekt beschreiben. So wird beispielsweise in Abbildung 4a) das zu erfassende Objekt mit dem 3D-Feature-Modell in Abbildung 4b) durch seine Eckpunkte beschrieben. Wie man schnell bemerkt, ist es schwer

(a) Objekt (b) 3D-Feature Model

Abbildung 4: Objekt und 3D-Feature Modell im Vergleich. Quelle: eigene Erstellung

vorstellbar, dass 4b) zu 4a) passt. Das liegt daran, dass zum einen die Pose nur geschätzt ist und zum anderen die Intrinsik der Kameras unterschiedlich ist. Näher auf die Kameraintrinsik wird in Kapitel 2.3.2 eingegangen.

2.2.1 Gewinnung aus 3D-Modell

Ein 3D-Feature-Modell kann anhand eines 3D-Modells gewonnen werden. Dies geschieht indem auf das 3D-Modell Filter aus der Bildverarbeitung angewandt werden. Da nur wenige Filter sich auf den dreidimensionalen Raum anwenden lassen, ist es erforderlich das Modell in den zweidimensionalen Raum zu bringen und es aus verschiedenen Posen zu rendern. Es ist auch möglich, direkt aus dem Datensatz des 3D-Modells Feature herauszulesen. Das bietet beispielsweise *VRML* an. Dieser Datentyp speichert unter anderem Eckpunkte, um das Modell zu beschreiben. Diese können ausgelesen und ohne weitere Verarbeitungsschritte verwendet werden.

2.2.2 Gewinnung aus Bildersequenz

Eine weitere Möglichkeit besteht darin, ein 3D-Feature-Modell aus einer Bildersequenz eines Objekts zu gewinnen. Dazu wird ein Objekt aus mehreren (sich überschneidenden) Bildern aufgenommen und in jedem Bild 2D-Feature mit einem Filter seiner Wahl gewonnen. Korrespondierende Feature zwischen verschiedenen Bildern werden gesucht und daraus die Kameraverschiebung geschätzt. Anhand dieser Informationen lassen sich die 2D-Feature mit einer zusätzlichen Tiefeninformation versehen und somit als 3D-Feature nutzen. Für weiterführende Informationen sei an dieser Stelle das mathematische Modell der *Epipolargeometrie* genannt. Sie stellt die geometrische Beziehung zwischen verschiedenen Kamerabildern des gleichen Objekts dar. Dieses Verfahren wird in Computer Vision auch als *Structure from Motion* (SfM) bezeichnet.

2.3 Mathematische Projektion/Transformation

Im Kontext der Computergrafik versteht man unter der mathematischen Projektion/Transformation die Abbildung eines 3D-Punktes, dort meistens Vertex genannt, auf eine 2D-Bildebene. Dies ist veranschaulicht in Abbildung 5. Anhand der intrinsischen und extrinsischen Parameter des Kame-

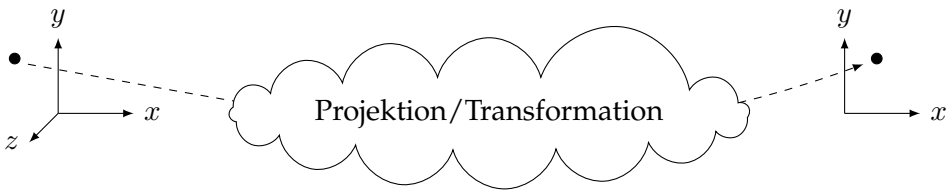

Abbildung 5: Mathematische Projektion/Transformation. Quelle: eigene Erstellung

ramodells kann die mathematische Projektion/Transformation beschrieben werden. Dabei werden die extrinsischen Parameter benötigt, um Vertices vom Weltkoordinatensystem in das Kamerakoordinatensystem umzurechnen (Transformation) und die intrinsischen Parameter vom Kamerakoordinatensystem in das Bildkoordinatensystem (Projektion).

2.3.1 Kameraextrinsik

Die Überführung eines 3D-Punktes in das Kamerakoordinatensystem wird durch die extrinsischen Parameter beschrieben. Sie enthalten die Translation und Rotation der Kamera relativ zum Weltkoordinatensystem. Die Kamera kann sich also in allen sechs Freiheitsgraden im Weltkoordinatensystem bewegen.
Die extrinsischen Parameter können mit

$$K_e = T \cdot R \tag{2.3.1}$$

beschrieben werden. Wobei T die Translation und R die Rotation angibt. Die Translationsmatrix lässt sich notieren als

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2.3.2}$$

wobei die vierte Spalte den Translationsvektor (t_x, t_y, t_z) und somit die Verschiebung angibt. Die Rotation kann notiert werden als

$$R = R_z(\gamma) \cdot R_y(\beta) \cdot R_x(\alpha), \tag{2.3.3}$$

wobei R_x, R_y, R_z die Rotation um die x-, y- und z-Achse angeben und jeweils eine 4×4 Matrix beschreiben.

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha & -sin\alpha & 0 \\ 0 & sin\alpha & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3.4}$$

$$R_y(\beta) = \begin{pmatrix} cos\alpha & 0 & sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -sin\alpha & 0 & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3.5}$$

$$R_z(\gamma) = \begin{pmatrix} cos\alpha & -sin\alpha & 0 & 0 \\ sin\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3.6}$$

Da Matrizenmultiplikationen nicht kommutativ sind, ist es sehr wichtig, dass man die Translationsmatrix und Rotationsmatrix nicht vertauscht. So ist $T \cdot R \neq R \cdot T$. Dies lässt sich anhand eines Beispiels zeigen. Sei $T = (2, 1, 0)$ und $R_x(90)$, so ergibt sich für $T \cdot R$

$$K_e = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 0 & -1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3.7}$$

und für $R \cdot T$

$$K_e = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3.8}$$

2.3.2 Kameraintrinsik

Um einen Punkt vom Kamerakoordinatensystem in das Bildkoordinatensystem zu überführen, wird mit den intrinsischen Parameter gearbeitet. Die intrinsischen Parameter sind unabhängig von den Äußeren und konstant. Die Parameter umfassen *Bildweite*, *Bildhauptpunkt* und *Scherungswinkel*.
Die Bildweite, oder auch focal length f, gibt den Abstand zwischen Projektionszentrum der Kamera und der Bildebene an, auf die die betrachtete

3D-Welt zweidimensional abgebildet werden soll. Die Bildebene ist dabei parallel zu der x, y-Ebene des 3D-Koordinatensystem. Sie besitzt ein eigenes lokales Koordinatensystem, dessen Ursprung in der oberen linken Ecke liegt. Die Position eines Punktes auf dieser Ebene wird in homogenen Koordinaten angegeben, die sich aus der Multiplikation mit der inversen Kameraextrinsik ergibt (2.3.3).
Der Bildhauptpunkt ist ein in Bildkoordinaten (x_c, y_c) angegebener Schnittpunkt der optischen Achse mit der Bildebene. Idealerweise liegt er im Bildmittelpunkt, wobei dann $x_c = y_c$ gilt, wenn wir von einem 1:1 Bildformat ausgehen.
Der Scherungswinkel oder auch $skew$-Parameter wird zwischen den Achsen mit s angegeben. Üblicherweise wird er für normale Kameras und Abbildungen mit 0 angenommen. Aus den beschriebenen Parametern ergibt sich folgende Matrix:

$$K_i = \begin{pmatrix} f & s & x_c \\ 0 & f & y_c \\ 0 & 0 & 1 \end{pmatrix} \tag{2.3.9}$$

In der Regel sind die Koordinatenachsen der Bildebene jeweils unterschiedlich und zumeist auch unabhängig vom 3D-Koordinatensystem der Kamera skaliert. Da der Bildsensor einer Kamera diskret aufgebaut ist und nicht immer quadratische Pixel hat, müssen bei der Abbildung vom Kamerakoordinatensystem in das Bildkoordinatensystem jeweils spezifische Skalierungsfaktoren der x- und y-Richtung berücksichtig werden. Seien m_x und m_y die Skalierungsfaktoren in beide Richtungen, die die Anzahl der Pixel pro Längeneinheit beschreiben. Daraus ergibt sich für f, die Brennweite in Abhängigkeit der Pixeldimensionen mit $a_x = f \cdot m_x$ und $a_y = f \cdot m_y$. Gleichermaßen ergibt sich für (x_0, y_0) abgeleitet aus den Pixeldimensionen $u_0 = x_0 \cdot m_x$ und $v_0 = y_0 \cdot m_y$.
Die nun angepasste Gleichung 2.3.9 ergibt sich als:

$$K_i = \begin{pmatrix} a_x & s & u_0 \\ 0 & a_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.3.10}$$

2.3.3 3D-2D-Abbildung

Zusammenfassend kann man mit der Kameraintrinsik und -extrinsik einen 3D-Punkt auf eine 2D-Bildkoordinate projizieren.
Es sei P_{3D} der 3D-Punkt und P'_{3D} seine 2D-Abbildung. Die Extrinsinc der Kamera ist beschrieben durch K_e (2.3.1). Damit ist zwar die Position der Kamera definiert, aber was benötigt wird, ist P im Kamerakoordinatensystem. Um diesem, der relativ zum Weltkoordinatensystem angegeben ist, im Kamerakoordinatensystem auszudrücken, muss dieser Punkt mit der Inversen von K_e multipliziert werden. Danach erfolgt die Umrechnung vom

Kamerakoordinatensystem ins Bildkoordinatensystem.
Den Schritt der mathematischen Projektion aus Abbildung 5 kann durch 2.3.11 verdeutlicht werden.

$$\overbrace{\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}}^{\text{Weltkoord.}} \underbrace{\longrightarrow}_{\text{extr. Param.}} \overbrace{\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}}^{\text{Kamerakoord.}} \underbrace{\longrightarrow}_{\text{intr. Param.}} \overbrace{\begin{pmatrix} x \\ y \end{pmatrix}}^{\text{Bildkoord.}} \tag{2.3.11}$$

Die Inverse von K_e ergibt sich aus der Multiplikation der inversen Translation und inversen Rotation. Für die inverse Rotation erhält man

$$R^{-1} = R_z(-\gamma) \cdot R_y(-\beta) \cdot R_x(-\alpha). \tag{2.3.12}$$

Es sind also einfach die Winkelangaben zu negieren. Die inverse Translation ist ähnlich trivial. Für sie gilt

$$T^{-1} = \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3.13}$$

Hier wird der Translationsvektor negiert $(-t_x, -t_y, -t_z)$. Damit ergibt sich als Inverse von K_e

$$K_e^{-1} = R^{-1} \cdot T^{-1} \tag{2.3.14}$$

Um P_{3D} auf P'_{3D} abzubilden ergibt sich nun folgende Formel

$$P'_{3D} = K_i \cdot K_e^{-1} \cdot P_{3D} \tag{2.3.15}$$

2.3.4 Euklid′scher Abstand

Da für die Bewertung einer Pose im späteren Verlauf der Arbeit der Abstand zwischen zwei Punkten berechnet werden soll, wird an dieser Stelle das Verfahren nach Euklid erläutert. Neben diesem wird in [6] noch die Manhattan- und Maximums-Distanz vorgestellt. Auf sie soll an dieser Stelle nur verwiesen sein.
Der *Euklid'sche Abstand* $d(p_1, p_2)$ kann auf zwei Punkte in einer Ebene, wie auch Raum, genutzt werden. Grundätzlich lässt sich diese Metrik auch auf jede beliebige Dimension anwenden. Im allgemeinen Fall des n- dimensionalen euklid′schen Raumes ist er für zwei Punkte oder Vektoren durch den Differenzvektor zwischen zwei Punkten definiert. Sind beide Punkte p_1 und p_2 durch die Koordinaten

$$p_1 = (k_1, k_2, ..., k_n)$$

und

$$p_2 = (l_1, l_2, ..., l_n)$$

gegeben, so ergibt sich folgende Gleichung:

$$d(p_1, p_2) = \sqrt{\sum_{i=1}^{n} (k_i - l_n)^2} \tag{2.3.16}$$

2.4 Posenschätzung

In den vorherigen Kapiteln wurden die Grundlagen der mathematischen Projektion erläutert. Anhand dieser und in diesem Kaptitel eingeführten Grundlagen ist es möglich aus Korrespondenzen die Kameraextrinsik zu schätzen. Dies kann mit Linien- und Punktkorrespondenzen geschehen sowie zwischen 2D-2D-und 3D-2D-Korrespondenzen. In dieser Arbeit soll sich allerdings auf 3D-2D-Korrespondenzen konzentriert werden.
Mit gegebenen n Korrespondenzen zwischen 3D-Koordinaten und 2D- Bildpunkten sollen die sechs Freiheitsgrade der Kamera geschätzt werden. Dies wird häufig als das *Perspective-n-Point* (PnP) Problem bezeichnet. Man unterscheidet zwei Methoden zur Posenschätzung. Zum einen lineare Methoden, wie der *DLT*[5] Algorithmus und zum anderen iterative Methoden.
Es lässt sich festhalten, dass eine Pose als gefunden zählt, wenn für alle n ein K_e gefunden wurde, sodass folgende Gleichung gilt:

$$\underbrace{\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix}}_{\text{2D-Bildpunkt}} = K_e \underbrace{\begin{pmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{pmatrix}}_{\text{3D-Koordinate}} \tag{2.4.1}$$

2.4.1 Lineare Methode

Lineare Methoden berechnen K_e durch das Aufstellen und Lösen eines linearen Gleichungssystem. Sie haben den Vorteil, dass sie keine Vorkenntnisse über eine intiale Pose benötigen. Allerdings besitzen sie eine hohe Komplexität und sind nur schnell für ein kleines n [14]. Das bekannteste Verfahren ist *Direct Linear Transformation* (DLT) und soll hier in den Grundzügen beschrieben werden. Zunächst wird, beispielsweise nach [1], die Glei-

[5] *Direct linear transformation*

chung 2.4.1 in die Form

$$x_i = \frac{K_{e11}X_i + K_{e12}Y_i + K_{e13}Z_i + K_{e14}}{K_{e31}X_i + K_{e32}Y_i + K_{e33}Z_i + K_{e34}}$$
$$y_i = \frac{K_{e21}X_i + K_{e22}Y_i + K_{e23}Z_i + K_{e24}}{K_{e31}X_i + K_{e32}Y_i + K_{e33}Z_i + K_{e34}} \tag{2.4.2}$$

überführt. Wobei K_{eab} für den Eintrag der K_e-Matrix an Zeile a und Spalte b steht. Anschließend wird nach [15] die Gleichungen aus 2.4.4 als folgende Gleichung

$$Ap = 0 \tag{2.4.3}$$

notiert. Hierbei ist p die Representation von K_e als Spaltenvektor und A eine $2n \times 12$ Matrix mit den aufgetragenen 2D und 3D-Punkten.

$$A = \begin{pmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 & -x_1 \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 & -y_1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -x_2X_2 & -x_2Y_2 & -x_2Z_2 & -x_2 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 & -y_2X_2 & -y_2Y_2 & -y_2Z_2 & -y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -x_nX_n & -x_nY_n & -x_nZ_n & -x_2 \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -y_nX_n & -y_nY_n & -y_nZ_n & -y_n \end{pmatrix} \tag{2.4.4}$$

A hat den Rang 11 und einen eindimensionalen Nullraum [16]. Daraus lässt sich schließen, dass sechs Korrespondenzen zur Lösung des Gleichungssystems benötigt werden. Die genaue Herleitung von A lässt sich [16] entnehmen.
Eine Gleichung der Form $Ap = 0$ kann mit Hilfe der *Singulärwertszerlegung* (SVD) gelöst werden. Da A bekannt ist, kann somit p berechnet werden. Dazu wird der Nullraum der Matrix A bestimmt.
Wie in [14] angegeben ist DLT nicht robust, was hauptsächlich damit begründet werden kann, dass die Korrespondenzen in der Regel nicht korrekt sind. Mit sogenannten *Robusten Schätzern* (*RANSAC*, *M-Schätzern*) können zwar fehlerhafte Korrespondenzen herausgefiltert werden. Aber schon eine falsche Korrespondenz erzeugt eine inkorrekte Pose.
Sofern die Korrespondenzen korrekt sind und, wie in unserem Fall, die Kameraintrinsik K_i bekannt ist, kann die Pose bis auf den Skalierungsfaktor geschätzt werden.
Lineare Methoden im Allgemeinen liefern keine genau Pose. Häufig werden diese Methoden dazu genutzt, eine grobe Pose zu schätzen und mit Hilfe einer iterativen Methode zu verfeinern.

2.4.1.1 POSIT

Beispielsweise benutzt *POSIT*[6] dieses Vorgehen. *POSIT* wurde 1995 von DeMenthon und Davis vorgestellt [17]. Im Grunde besteht dieser Algorithmus aus zwei Schritten.

[6] Pose from Orthography and Scaling with iterations

Zu Beginn wird mit einer linearen Methode eine initiale Pose geschätzt (*POS*) und diese im nächsten Schritt mit einer iterativen Methode verfeinert (*IT*). Laut den Autoren reichen im Durchschnitt vier bis fünf Iterationen aus, damit der zweite Schritt konvergiert. Das Problem dieses Algorithmus ist, dass er nicht angewandt werden kann, wenn Korrespondenzen koplanar sind. In [18] wird ein ähnlicher Ansatz beschrieben, der koplanare Korrespondenzen handhaben kann.

2.4.2 Iterative Methode

Iterative Methoden haben den Vorteil, dass sie eine genau Pose liefern. Allerdings muss ein Initialpose K_e^0 gegeben sein. Diese wird iterativ verbessert. Wie in 2.4.1.1 angegeben, reichen unter Verwendung des POSIT Algorithmus vier bis fünf Iterationen.
Als ein wichtiges Maß zur Bewertung der neuen Pose dient hier eine Fehlerfunktion. Umso höher der Wert der Fehlerfunktion ist, desto schlechter ist die neue Pose. Ziel ist es, diesen Wert zu minimieren. In allen Ansätzen [19, 20, 17] wird eine Fehlerfunktion anhand der geschätzten Pose definiert. Dazu wird zumeist der quadrierte Projektionsfehler als Fehlerfunktion genutzt:

$$[R|T] = \sum_{i=0}^{n} (f(P_i) - p_i)^2 \tag{2.4.5}$$

Hierbei ist $f(P_i)$ eine Funktion, die einen 3D-Punkt (P_i) in Abhängigkeit zu $[R|T]$ (Rotation und Translation) auf eine 2D-Bildkoordinate abbildet (siehe 2.3) und n die Anzahl der Korrespondenzen angibt. Neben dieser Fehlerfunktion können natürlich auch andere verwendet werden. Die extrinsischen Kameraparameter, die für $f(P_i)$ erforderlich sind, werden anhand dem numerischen Verfahren der nichtlinearen Optimierung 2.4.2.1 berechnet.

2.4.2.1 Nichtlineare Optimierung

Um die Pose für die nächste Iteration zu schätzen, werden sogenannte nichtlineare Optimierer genutzt. Diese zählen zu numerischen Verfahren und können nur lokale Minima/Maxima (siehe 2.5) bestimmen [21]. Daher ist eine initiale Pose notwendig, um nicht ein lokales Extrem zu optimieren. Wie in 2.4.1 bereits erwähnt, wird dazu häufig zunächst mit einer linearen Methode diese geschätzt und anschließend verbessert. Δ_i ist die Differenz, um die die neue Pose K_e^{i+1} aktualisiert wird. Daraus folgt

$$K_e^{i+1} = K_e^i + \Delta_i. \tag{2.4.6}$$

Mit der nichtlinearen Methode kann Δ berechnet werden, welches von Iteration zu Iteration den Fehler, beispielsweise wie in 2.4.5 berechnet, verrin-

gert.
Die Methoden kann man in drei Gruppen einteilen. Die erste Gruppe beinhaltet die *Ableitungsfreien Methoden*. Darunter fallen die *Intervallhalbierungsverfahren* und *Downhill-Simplex-Verfahren* [22]. Diese Methoden benötigen viele Iterationen. Sie sind aber robust und benötigen keine aufwendige Berechnung eines Gradienten, wie es das *Gradientenverfahren* aus der zweiten Gruppe verlangt. Neben diesem Verfahren kann man noch das *Sekantenverfahren* und das *Quasi-Newton-Verfahren*[23] in die Gruppe der *ersten Ableitung* einordnen. Die dritte Gruppe beschreibt die Methoden, die die *zweiten Ableitungen* benötigen. Darunter fällt der *Gauss-Newton*-Verfahren, der *Levenberg-Marquardt*-Algortihmus [24] und die *Newton*-Verfahren.
Diese Verfahren seien an dieser Stelle erwähnt. Für weitere Informationen wird auf [14] verwiesen.

2.5 Optimierungsverfahren

"...Bei Optimierungsproblemen sind das Modell und die gewünschten Ausgaben bekannt, allerdings sind die Eingaben, die zu den gewünschten Ausgaben führen unbekannt. Das Ziel bei Optimierungsproblemen ist es, solche Eingabewerte zu finden, mit denen das System die gewünschten Ausgaben berechnet..."[25]

Optimierungsverfahren werden dort angewandt, wo der Suchraum zu groß ist, um alle Möglichkeiten auszuprobieren (*Brute-Force*). In unserem Beispiel haben wir es mit sechs Freiheitsgraden zu tun. Dementsprechend würde folgende Abbildung nicht zwei Achsen, sondern sechs Achsen besitzen, also einen sechsdimensionalen Suchraum beschreiben. Zur besseren Veranschaulichung wird sich an dieser Stelle auf einen eindimensionalen Suchraum beschränkt.
Ziel ist es, das Minimum der Funktion $f(x)$ zu finden. Analog zu der Aufgabenstellung dieser Arbeit kann man annehmen, dass $f(x)$ versucht die extrinsik zu finden, die einen minimalen Pixelabstand zwischen 2D Feature und einem projiziertem 3D Feature Modell erzeugt. Näheres dazu kann Kapitel 4 entnommen werden.
Es handelt sich um ein sogenanntes Minimierungsproblem. Es wird der kleinste Wert der Funktion $f(x)$ gesucht. Die Suche nach dem $Maximum$ für $f(x)$ wird als Maximierungsproblem bezeichnet. Alle weiteren Angaben, die sich auf das *Minimum* beziehen, sind analog für das *Maximum* zu verstehen.
Das *Minimum* lässt sich weiter unterteilen: zum einen in ein *lokales Minimum* und zum anderen in ein *globales Minimum*. Das *lokale Minimum* beschreibt, wie der Name vermuten lässt, einen Funktionswert, der in seiner direkten Nachbarschaft minimal ist. Also nicht über das gesamte Intervall,

$$\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}, \tag{2.5.1}$$

sondern über ein Teilintervall, wie beispielweise

$$\{x \in \mathbb{R} \mid 0,3 \leq x \leq 0,5\}. \tag{2.5.2}$$

Weiter in ein *globales Minimum*. Dieses gilt es in den meisten Fällen zu finden und beschreibt das *Minimum* über das gesamte Intervall (2.5.1). Es ist definiert als das *Minimum* aller *lokalen Minima* (2.5.3).

$$\min(min_{lokal1}, min_{lokal2}, ..., min_{lokaln}) \tag{2.5.3}$$

Das Finden des *globalen Minimums* stellt sich meistens als eine Herausforderung dar. Die meisten Optimierungsverfahren (*Partikelfilter*, *Simulated Anealing*, *Genetischer Algorithmus*, ...) besitzen extra Schritte, um nicht in ein lokales Minimum zu "fallen".
"Fallen"beschreibt das Phänomen, dass der Algorithmus ohne Anpassung an dieses Problem nur mit einer geringen Wahrscheinlichkeit von diesem Lösungskanditaten ablässt und ihn auch bis zur Terminierung des Algorithmus beibehält.

2.5.1 Simulated Anealing

Simulated Anealing[7] beschreibt ein Verfahren, das sich mit dem Schmieden von Eisen vergleichen lässt. Zu Beginn ist das Eisen am wärmsten und lässt sich gut verformen. Die grobe Form des Eisens wird erzeugt. Im übertragenden Sinn würde es das Zulassen großer Parameteränderungen im Suchraum bedeuten. Im Laufe der Verarbeitung des Materials kühlt es ab und lässt sich immer weniger verändern. Es können nur noch Änderungen an den Feinheiten vorgenommen werden. Dieses Verhalten wird durch kleinere Parameteränderungen beschrieben. Bei *Simulated Anealing* wird sich erhofft, dass zu Beginn der Laufzeit große Parameteränderungen zugelassen werden und die lokalen Extrema übersprungen werden. Im weiteren Verlauf, wenn nur noch kleinere Änderungen zugelassen werden, soll die Lösung optimiert werden.

2.5.2 Partikelfilter

Partikelfilter oder auch *Partikelschwarmoptimierung* (PSO) wurde 1995 von Russell Eberhart und James Kennedy eingeführt [26, 27]. PSO ist eine Weiterentwicklung der früheren Schwarmalgorithmen. Jeder Partikel p eines Schwarms S stellt einen Lösungskandidaten dar, der zur Initialisierung zufällig im Suchraum verteilt wird. In weiteren Schritten wird versucht, jeden Partikel zu optimieren.

[7] Simulated Anealing ≈ Simuliertes Abkühlen

Ein Partikel besitzt eine Geschwindigkeit $\overrightarrow{v_t}$ zur Zeit t, die ihn durch den Suchraum bewegt, solange bis er das Optimum gefunden hat oder ein anderes Abbruchkriterium eingetreten ist. Anschaulich kann man das anhand eines zweidimensionalen Suchraums beschreiben, wie in Abbildung 6 zu sehen. Bei jeder Iteration nähert sich S dem Optimum an. Jeder schwarze

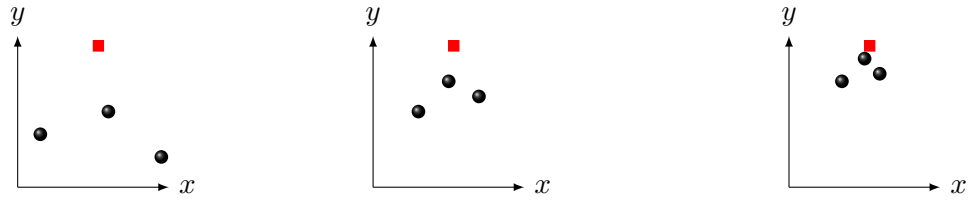

Abbildung 6: Drei Iterationen eines Partikelfilters. Quelle: eigene Erstellung

Punkt definiert einen Partikel. Das rote Quadrat beschreibt das Optimum, das es zu finden gilt. $\overrightarrow{v_t}$ setzt sich aus insgesamt drei Komponenten zusammen:

$$\overrightarrow{v_t} = \underbrace{\overrightarrow{v_{t-1}}}_{\text{träge}} + \underbrace{\overrightarrow{v_k}}_{\text{kognitiv}} + \underbrace{\overrightarrow{v_s}}_{\text{sozial}}. \tag{2.5.4}$$

- $\overrightarrow{v_{t-1}}$ ist definiert als die Geschwindigkeit aus dem letzten Iterationsschritt und kann aus ihm übernommen werden, auch bekannt als Inertial, träge und Gewicht von p. Für $t = 1$ werden die Geschwindigkeiten zufällig initialisiert.
- $\overrightarrow{v_k}$ beschreibt die kognitive Komponente. Jeder Partikel speichert sich seine beste Position. Diese Position fließt in jeden weiteren Iterationsschritt zur Berechnung von v_t mit ein. Damit wird beschrieben, dass ein Partikel sich von seinem Optimum zwar entfernen kann, aber eine gewisse Tendenz zu ihm aufweist.
- $\overrightarrow{v_s}$ wird als die soziale Komponente bezeichnet. Dieser Geschwindigkeitsvektor zeigt zudem p, dass in S die beste Fitness hat p_{best}.

Um nicht in einem lokalen Minimum/Maximum hängen zu bleiben, wird Formel 2.5.4 an zwei Stellen erweitert. Zum einen wird die kognitive, als auch die soziale Komponente jeweils mit einer zufälligen Geschwindigkeit erweitert. Dadurch ist gewährleistet, dass ein Partikel sich nicht auf direktem Weg p_{best} nähert und dadurch auch noch andere, eventuell bessere Lösungen findet. Des weiteren wird jede der drei Komponenten mit einer Konstante gewichtet. Dadurch erhält man drei Parameter w_t, w_k, w_s, mit denen man die drei Geschwindigkeitskomponenten unterschiedlich

gewichten kann. Die aktualisierte Formel lautet dann

$$\overrightarrow{v_{t+1}} = \underbrace{w_t \cdot \overrightarrow{v_t}}_{\text{träge}} + \underbrace{w_k \cdot (\overrightarrow{v_k} + \overrightarrow{v_u})}_{\text{kognitiv}} + \underbrace{w_s \cdot (\overrightarrow{v_s} + \overrightarrow{v_v})}_{\text{sozial}} \tag{2.5.5}$$

In [28] wird für $w_t = 0,8$, $w_k = 2$ und $w_s = 2$ empfohlen. Weiter wird angegeben, dass $\overrightarrow{v_u}$ und $\overrightarrow{v_v}$ in jedem Iterationsschritt für jeden Partikel neu berechnet werden und die Elemente der Vektoren einen Wert aus $[0, 1]$ besitzen. Abbildung 7 soll 2.5.4 bildlich darstellen. Es sei angemerkt, dass der schwarze Kreis im Zentrum wieder einen Partikel an seiner Position zu $t-1$ darstellt.

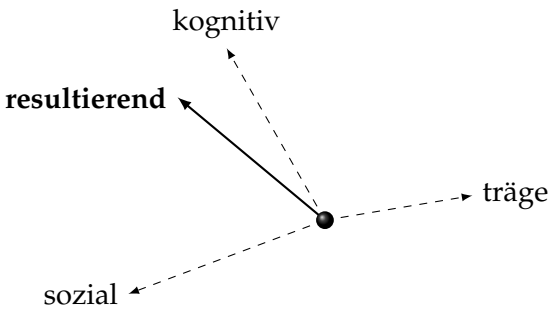

Abbildung 7: Grafische Darstellung v_t. Quelle: eigene Erstellung

Ein Nachteil der Partikelfilter ist ihr kontinuierliches Verhalten der Optimierung. Das zu optimierende Problem muss einen Suchraum besitzen, den man kleinschrittig durchlaufen kann. Besteht er aus zu großen Schritten, ist ein *Partikelfilter* nicht das Mittel der Wahl. Der Grund dafür ist, dass ein Partikel eine Geschwindigkeit besitzt und bewegt sich im Optimalfall kontinuierlich durch den Suchraum. Besitzt ein Suchraum Sprünge zwischen Parametern, so kann keine eindeutige Bewegungsrichtung angegeben werden. Der Suchraum des Lösungsvorschlags dieser Arbeit besitzt Parametersprünge, die sogar so groß sind, dass von einem *Partikelfilter* zur Optimierung abgesehen wurde.
Anders als *Partikelfilter* lassen sich *genetische Algorithmen* gut auf einen Suchraum mit großen Parametersprüngen anwenden.

2.5.3 Genetische Algorithmen

Genetische Algorithmen (GA) sind Optimierungsverfahren, die sich am Vorbild der biologischen Evolution orientieren. Erstmals wurden sie Anfang der 60er Jahre von John Holland vorgestellt [29]. Sie gehören den *Evolutionären Algorithmen* (EA) an und neben ihnen existieren drei weitere Teilgebiete:

- Evolutionäre Programmierung (EP)
- Genetische Programmierung (GP)
- Evolutionsstrategien (ES)

Die Unterteilung ist hauptsächlich geschichtlich bedingt, da sie sehr hohe Ähnlichkeiten aufweisen. Trotzdem existieren zwischen den Teilgebieten gewisse Unterschiede.
In *genetischen Algorithmen* werden Individuen klassisch in Bitstrings kodiert. In *Evolutionsstrategien* und in der *evolutionären Programmierung* zumeist in reelen Vektoren.
Ein weiterer markanter Unterschied besteht in der Selektion der Individuen für die nächste Generation. *Genetische Algorithmen* und *Evolutionäre Programmierung* nutzen eine fitnesspropotionale Selektion auf stochastischer Basis. Auch Individuen mit einer schlechten Fitness können, wenn auch nur mit einer sehr geringen Wahrscheinlichkeit, in die nächste Generation übernommen werden. Kapitel 4.4.5 bietet dazu weitere Informationen. Bei *Evolutionsstrategien* dagegen wird eine deterministische Selektion durchgeführt. Es werden also nur die Individuen mit der besten Fitness in die nächste Generation übernommen.

2.5.3.1 Biologische Grundlagen

Lebewesen tragen in jeder einzelnen Zelle Erbinformation, die sich im Laufe des Lebens nicht verändert.
Pflanzen sich zwei Lebewesen fort, entstehen neue Lebewesen, mit neuen Erbinformationen. Die Erbinformationen der Eltern werden miteinander kombiniert und zu neuen zusammengesetzt. Aus diesem Grund sind Kinder ihren Eltern ähnlich, aber nicht identisch.
Während dem Schritt der Fortpflanzung können spontane Änderungen der Erbinformationen auftreten, die als Ergebnis ebenfalls zu neuen Erbinformationen führen. Solche Änderungen werden beispielsweise durch äußere Umwelteinflüsse verursacht.
Abschließend sei das Evolutionsprinzip *Survival of the fittest* nach Charles Darwin erwähnt. Es besagt, dass diejenigen Lebewesen einer Population überleben, die sich am besten an die Umwelt angepasst haben. Im Laufe der Evolution bilden diese Lebewesen die größte Zahl an Nachkommen und beeinflussen die weitere Evolution der Population am stärksten. Dieses Phänomen wird auch als *natürliche Selektion* bezeichnet. Die Natur trifft eine Auswahl an Lebewesen, deren Nachkommen immer besser an die Umwelt angepasst sind.

Allel, Genotyp und Phänotyp

Ein Allel[8] ist eine konkrete Ausprägung eines Gens, das sich an einem Ort eines Chromosoms befindet. Wird das Gen als eine Variable aufgefasst, dann kann man ein Allel als den Wert der Variable auffassen. Der Genotyp[9] beschreibt die Gesamtheit aller Gene eines Individuums. Ein Phänotyp hingegen ist das Erscheinungsbild und beschreibt die Menge aller Merkmale eines Individuums. Er ist also der ausgeprägte Genotyp.

Individuum und Chromosom

Ein Individuum[10] im biologischen Kontext ist ein lebender Organismus (Lebewesen), dessen Gene, somit auch Erbinformationen, in einer Menge von Chromosomen[11] gespeichert sind. In den meisten Anwendungsfällen eines genetischen Algorithmus wird kein Unterschied zwischen einem Individuum und Chromosom gemacht, da es in der Regel ein Problem zu optimieren gilt, das man anhand eines Chromosoms beschreiben kann. Genaugenommen muss auch kein Unterschied gemacht werden, wenn ein Individuum aus nur einem Chromosom besteht. Ist das zu optimierende Problem auf mehrere Chromosome verteilt, ist die Unterscheidung erforderlich, denn ein Individuum besteht aus einer Anzahl von Chromosomen.

Population und Generation

Eine Menge von gleichartigen Individuen einer bestimmten Art wird als Population bezeichnet, sofern sie eine Fortpflanzungsgemeinschaft bilden und zur gleichen Zeit in einem einheitlichen Areal zu finden sind. Sterben Individuen einer Art oder werden geboren, verändert sich zwangsläufig auch die Größe der Population. Betrachtet man eine Population zu einem Zeitpunkt t, spricht man von der Generation$_t$.

2.5.3.2 Grundsätzlicher Ablauf

Abbildung 8 verdeutlicht den Ablauf eines genetischen Algorithmus anhand eines Flussdiagramms. Nach der Initialisierung durchläuft der Algorithmus eine Schleife, in der er die vier Schritte Fitnessberechnung, Selektion, Crossover und Mutation auf der Generation$_t$ durchführt. t wird bei jedem Durchlauf um eins erhöht.

[8] gr. einander
[9] gr. genos - Gattung und typos - Abbild
[10] lat. individuum = Unteilbares
[11] gr. *Farbkörper*

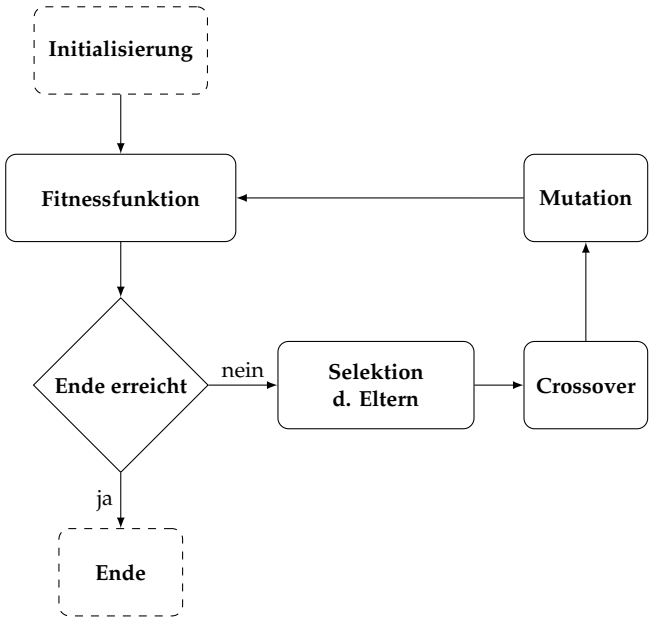

Abbildung 8: Einfaches Flussdiagramm eines genetischen Algorithmus. Quelle: eigene Erstellung

2.5.3.3 Wahrscheinlichkeits- und Fitnessfunktion

Anhand der Fitnessfunktion wird ein Individuum bezüglich der Problemstellung bewertet. Auf Grund der Aufgabenstellung ist die Fitness F besser zu bewerten, wenn die gefundene Pose besser wird. Eine Fitnessfunktion bildet alle Individuen auf ihre Fitness ab. In Abbildung 9 wurde eine Population zum Zeitpunkt $t = 1$ dargestellt. Das beste Individuum wurde rot markiert. Wichtig ist es auf allen Individuen dieselbe Metrik anzuwenden, um sie untereinander vergleichen zu können und auch eine Wahrscheinlichkeitsfunktion Wsk_i zu definieren. Sie ist gegeben als

$$Wsk_i = \frac{F_i}{\sum_{j=1}^{n} F_j} \tag{2.5.6}$$

wobei F_i die Fitness des aktuellen Individuums darstellt und der Nenner die aufsummierte Fitness aller Individuen darstellt. Wsk_i gibt die Wahrscheinlichkeit in Proportionalität zur Population an. Dabei gilt immer Gleichung 2.5.7.

$$\sum_{i=1}^{n} Wsk_i = 1 \tag{2.5.7}$$

Diese Eigenschaft wird für die fitnessproportionale Selektion (2.5.3.5) sehr nützlich sein.

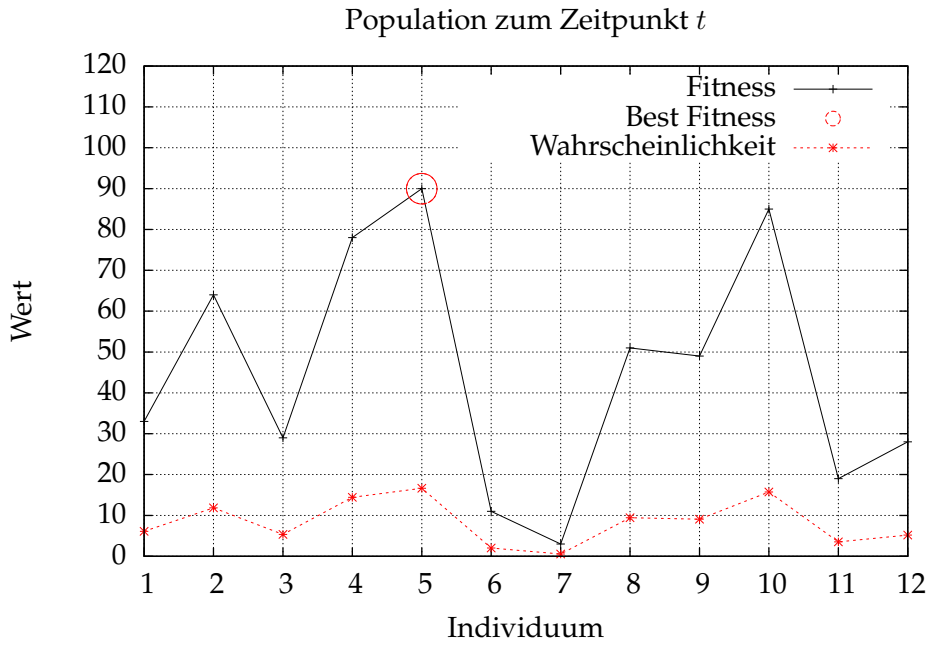

Abbildung 9: Beispielhafte Fitness- und Wahrscheinlichkeitsfunktion. Quelle: eigene Erstellung

2.5.3.4 Selektion

Der Selektion[12] ist immer eine Fitnessfunktion vorangestellt. Sie wählt Individuen, die zur Reproduktion zugelassen werden, aus. Je besser die Fitness eines Individuums ist, umso höher ist die Wahrscheinlichkeit selektiert und als ein Elternindividuum für die nächste Generation bestimmt zu werden. Die Wahrscheinlichkeit einer Selektion des Individuums ist gegeben aus Formel 2.5.6. Bei dieser Art der Selektion spricht man von einer fitnessproportionalen Selektion (Kapitel 2.5.3.5). Ein Nachteil ist ihr Dominanzproblem, da das gleiche Individuum gegebenenfalls mehrmals selektiert werden kann. Im anderem Extrem kann es auch passieren, dass das beste Individuum nicht mit in die nächste Generation übernommen wird. Weitere Verfahren sind *Rangbasierte Selektion* und *Turnierselektion*.

- *Rangbasierte Selektion* beschreibt ein Verfahren, bei dem die Individuen anhand der Fitness absteigend sortiert werden. Die Selektionswahrscheinlichkeit steigt, je besser der Rang eines Individuums ist. Der Vorteil ist, dass die Selektionswahrscheinlichkeit nicht mehr im direkten Verhältnis zur Fitness steht (Dominanzproblem).[30]

[12] lat. selectio 'Auswahl/Auslese'

- *Turnierselektion* ist eine weitere Selektionsart, die versucht das Dominanzproblem zu verhindern.[30] Um ein Individuum für die nächste Generation zu selektieren, werden k Individuen gezogen. Wobei $2 < k < n$ gilt. Zwischen ihnen wird im übertragenen Sinne ein Turnier ausgetragen. Das Individuum mit der besten Fitness gewinnt und wird in die nächste Generation übernommen. Dieses Vorgehen wird n-mal wiederholt, bis die Population$_{t+1}$ die selbe Anzahl an Individuen wie Population$_t$ besitzt. Allerdings kann es auch bei diesem Verfahren zu einem Dominanzproblem führen.

Das Dominanzproblem beschreibt das Phänomen, dass ein gutes Individuum spätere Generationen erheblich beeinflusst. Dies geschieht, indem der Genpool hauptsächlich aus seinen Erbinformationen besteht. Ein Problem für weitere Generationen besteht darin, dass nur noch ähnliche Individuen zu dem Dominierenden hervorgebracht werden. Dieses Phänomen ist dafür verantwortlich, dass ein genetischer Algorithmus in einem lokalen *Maximum/Minimum* hängen bleiben kann.

2.5.3.5 Fitnessproportionale Selektion

Da die fitnessproportionale Selektion ein häufig verwendetes Schema ist, soll sie hier näher beschrieben werden. Wie in 2.5.3.3 bereits erwähnt, wird jedes Individuum durch seine Fitness bewertet. Je besser die Fitness ist, desto wahrscheinlicher ist es, dass es in die nächste Generation aufgenommen wird. Eventuell sogar mehrfach.
Um das Prinzip der fitnessproportionalen Selektion zu erfüllen, hat sich als bekannteste Verfahren die *Glücksradauswahl*[13] durchgesetzt. Dabei wird jedes Individuum proportional zu seiner Wahrscheinlichkeit (2.5.3.3) auf eine Drehscheibe aufgetragen. Je höher die Wahrscheinlichkeit eines Individuums ist, desto mehr Platz nimmt es auf dem Glücksrad ein. Da die Summe der Wahrscheinlichkeiten aller Individuen immer 1 ergibt kann man den zugeordneten Platz auf der Drehscheibe direkt aus den Wahrscheinlichkeiten Wsk_i ablesen. Existiert nur ein Individuum mit einem Wsk von 1, dann füllt es $360°$ der Drehscheibe aus. Wie viel Grad ein Individuum einnimmt, lässt sich über Formel 2.5.8 einfach berechnen.

$$\alpha_i = Wsk_i \cdot 360° \tag{2.5.8}$$

Hier gilt, dass alle aufsummierten Winkel $360°$ ergeben, also alle Individuen die komplette Scheibe ausfüllen:

$$\sum_{i=1}^{n} \alpha_i = 360° \tag{2.5.9}$$

[13] auch bekannt als *roulette wheel selection*

Abbildung 10 verdeutlicht den Aufbau einer *Glücksradauswahl* und weist den Individuen gemäß ihrer Fitness F_i einen Platz auf der Scheibe zu. Folgender Dreipunktealgorithmus beschreibt das Vorgehen einer fitnessproportionalen Selektion.

- *Initialisierung:* Weise den Individuen fitnessproportional Platz auf dem Glücksrad zu
- *Auswahl eines Individuums*
  - Drehe das Glücksrad
  - Wähle das Individuum, dessen Sektor an der Markierung liegt.
- *Folgepopulation auswählen:* Führe Schritt - *Auswahl eines Individuums* - so oft durch, wie es Individuen in der Population gibt.

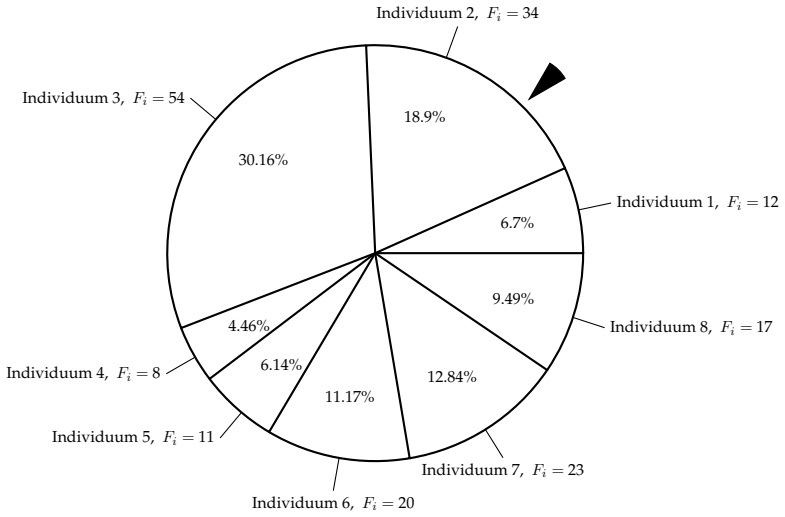

Abbildung 10: Glücksradauswahl der fitnessproportionalen Selektion. Quelle: eigene Erstellung

Besitzt ein Individuum eine hohe Fitness, so ist die Wahrscheinlichkeit höher in die nächste Generation aufgenommen zu werden. Da ein bereits selektiertes Individuum nicht vom Glücksrad genommen wird, besteht auch die Möglichkeit, dass ein Individuum mehrfach in die nächste Generation aufgenommen wird.

2.5.3.6 Crossover

Crossover oder auch Rekombination ist das Hauptwerkzeug der Evolution, mit dem neue Informationen in eine Population gebracht werden. Dazu

werden zwei Individuen ausgewählt, ihre Erbinformationen an einer Stelle aufgetrennt und neukombiniert zusammengefügt. Die Wahl der Individuen, wie auch die Stelle des Crossover-Punktes, wird per Zufall ausgewählt. Es existieren unterschiedliche Arten Crossover durchzuführen. Die bekanntesten Methoden sind *One-Point-Crossover*, *N-Point-Crossover*, *Template - Crossover*, *Uniform-Crossover* und *Shuffle-Crossover*. Bei dem *One-Point-Crossover* werden die Chromosome zufällig an einer Stelle aufgetrennt und über Kreuz mit den Partnerchromosomen zusammengefügt. Abbildung 11 verdeutlicht das. *Template-Crossover* erstellt zu Beginn eine binäre Schablone, die

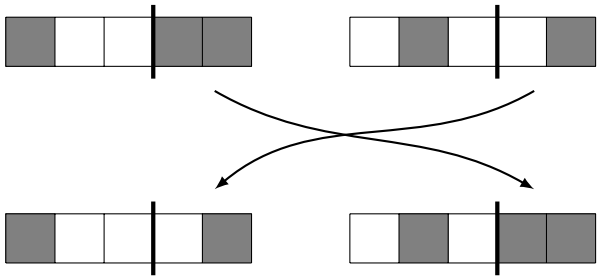

Abbildung 11: One-Point-Crossover. Quelle: eigene Erstellung

auf beide Individuen gelegt wird. Das erste Individuum erhält alle Allele von sich selbst, bei denen die Schablone eine Eins (in Abbildung 12 weiß dargestellt) hat. Die restlichen Allele werden von Individuum Zwei ausgewählt, wo die Schablone eine Null (in Abbildung 12 grau dargestellt) hat. Für das zweite Individuum ist das Verfahren ähnlich, Unterschied ist, dass die eigenen Allele mit einer Null und die fremden Allele mit einer Eins ausgewählt werden.

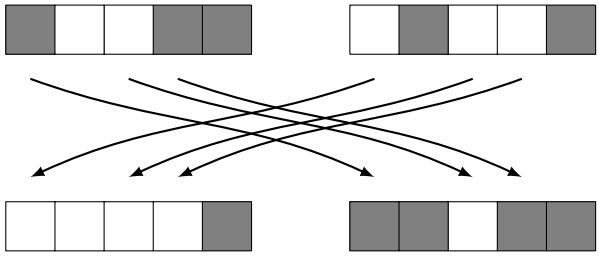

Abbildung 12: Template-Crossover. Quelle: eigene Erstellung

2.5.3.7 Mutation

Das zweite wichtige Werkzeug der Evolution ist die Mutation. Sie tritt wesentlich seltener ein als das Crossover und dient dazu, neues oder ver-

lorengegangenes Erbgut in die Population zu bringen. Tritt sie zu häufig ein wirkt sie sich negativ aus. Da die Evolution nicht mehr in eine Richtung hin optimiert wird, sondern zufällig neue Individuen hervorbringt. Man unterscheidet zwischen verschiedenen Mutationsarten. Zwei von ihnen sind zum einen die *Flipmutation* oder auch *Punktmutation* (Abbildung 13). Es wird zufällig ein Gen ausgewählt und negiert. Zum anderen die *Swapmutation* (Abbildung 14). Bei ihr werden zufällig gewählte Gene vertauscht.

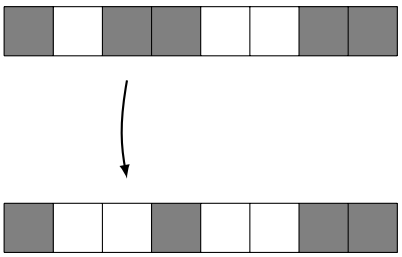

Abbildung 13: Punkt-Mutation. Quelle: eigene Erstellung

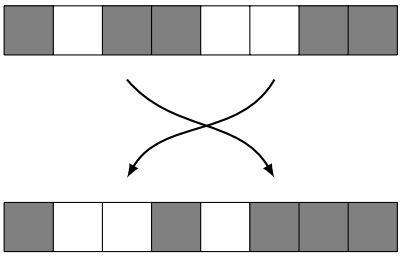

Abbildung 14: Swap-Mutation. Quelle: eigene Erstellung

2.5.3.8 Elitismus

In diesem Kapitel wird das erste Verfahren vorgestellt, das über den klassischen *genetischen Algorithmus* hinaus geht und ihn erweitert. Außerdem wird eine Option angegeben, die das Verlieren eines guten Kandidaten durch Selektion, Crossover oder Mutation verhindert.

Von *Elitismus* wird dann gesprochen, wenn das beste Individuum oder die k besten Individuen, wobei $1 < k < n$, in die nächste Generation automatisch übernommen werden. Obwohl ihre Wahrscheinlichkeit bei der fitnessproportionalen Selektion sowieso sehr hoch ist, steigt sie durch dieses Verfahren noch einmal. Trotz einer solchen Selektion ist es nicht garantiert, dass das beste Individuum unverändert übernommen wird.

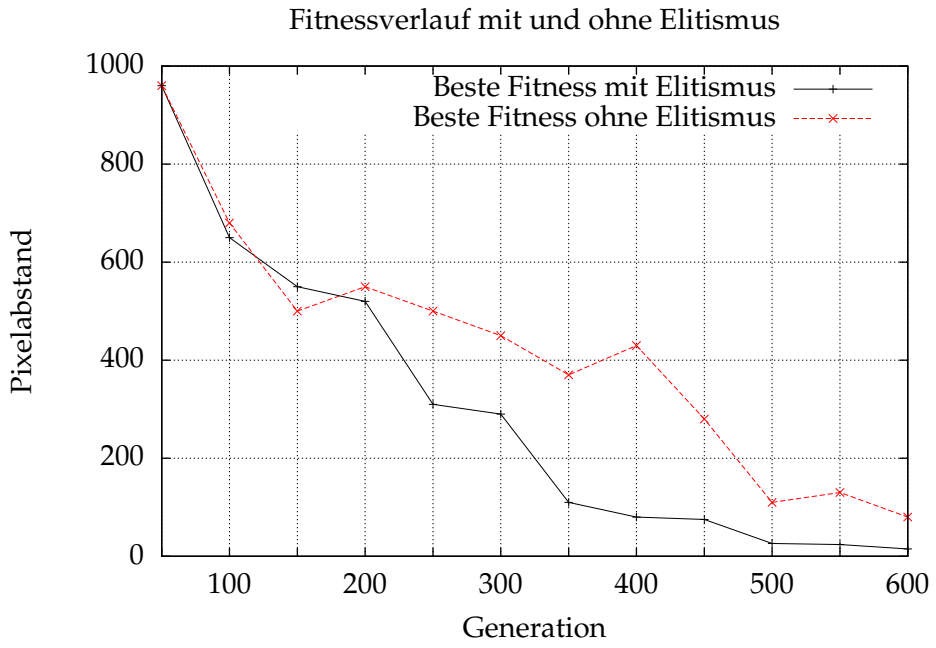

Abbildung 15: Fitnessverlauf mit und ohne Elitismus. Quelle: eigene Erstellung

Abbildung 15 zeigt den Fitnessverlauf des besten Individuums mit und ohne *Elitismus*. Mit *Elitismus* aber konvergiert der Algorithmus zu Beginn schnell und gegen Ende immer langsamer stetig gegen das Optimum. Ohne *Elitismus* geht das beste Individuum wenige Male verloren, woraus sich die Sprünge im Kurvenverlauf begründen. Eingeführt wurde die Elitismusoption von Goldberg 1989 [31].

2.5.3.9 Insel Evolution

Die *Insel Evolution* oder auch *island model* ist der erste Ansatz, mit dem verhindert werden soll, dass man in einem lokalen Extremum hängen bleibt. Sie wurde 1987 von Cohoon et al. [32] eingeführt. Der Ansatz basiert darauf, verschiedene Populationen auf einer Anzahl von Inseln K entwickeln zu lassen. Von Zeit zu Zeit migrieren Individuen von einer zu der anderen Insel (siehe Kapitel 3.3.1). Die Zahl der Migranten kann variieren, wird aber in [33] mit sechs angegeben. Mit einer Migration soll eine Katastrophe oder eine zufällige Änderung der Umweltbedingung simuliert werden. Es wird der Genpool vergößert und es können neue Lösungskandidaten entstehen, die sonst nur durch eine Mutation möglich gewesen wären.
Lässt man den Schritt der Migration aus, erhält man dasselbe Ergebnis, als würde man den *genetischen Algorithmus* K-mal laufen lassen. Neben der

Vergrößerung des Genpools ist ein entscheidender Vorteil der *Insel Evolution* die Parallelisierbarkeit. Jede Insel kann separat ausgeführt werden und nach jeder Generation wird das beste Individuum aller Inseln als momentan ideale Lösung angenommen.

2.6 VRML

VRML steht für *Virtual Reality Modeling Language* und ist eine 1995 eingeführte Beschreibungssprache für 3D-Szenen, Geometrie, Lichtquellen, Animationen, Interaktionsmöglichkeiten und Geräuschquellen. *VRML* gilt als Vorläufer von *X3D*, einer auf XML basierter Beschreibungssprache.
VRML-Dateien sind in *ASCII* bzw. *UTF-8* geschrieben und können mit einem einfachen Texteditor erzeugt werden. Zu erkennen ist eine *VRML*-Datei an dem postfix *.wrl* und kann von den meisten 3D - Modellierungswerkzeugen (*Blender*, *Maya*) importiert und auch exportiert werden.

```
1 #VRML V2.0 utf8

3 Shape {
4 appearance Appearance {
5 material Material {}
6 }
7 geometry Box {}
8 }
```

Listing 1: Hallo Welt in VRML

Der Aufbau besteht aus einer Auswertungszeile, die angibt, um welches Format es sich handelt. Zu sehen in 9 in Zeile 1. Danach folgt der eigentliche Inhalt als Objekte, die in *VRML* als Knoten bezeichnet werden. Der Hauptknoten ist in 9 *Shape*, der für sichtbare Objekte zuständig ist. An diesen Knoten sind zwei weitere Knoten angehangen. Zum einen *appearance*, der das Erscheinungsbild definiert und zum anderen *geometry*, der die Form des Objekts angibt. In *VRML* ist es möglich auf vorgefertigte Attribute zuzugreifen. So wird für den *geometry*-Knoten das vordefinierte Objekt *Box* genutzt und für die Erscheinung wird die Einstellung *Appearance* ausgewählt und in einem angehangenen Subknoten *material* das vordefinierte *Material* angegeben. Neben diesen existieren noch Knoten für Materialeigenschaften, Lichtquellen, Sensoren, Skriptknoten, Billboards, Transformationen und weitere.

2.7 Instant Vision

Das für diese Arbeit genutzte Framework ist die *Instant Vision*. Sie ist ein Teil der *Instant Reality* und ist ein vom *Fraunhofer IGD* entwickeltes Softwarepaket für *Virtual Reality* (VR) und *Augmented Reality* (AR). Die Stärke des Frameworks besteht darin, dass es Cross-Pattform angeboten wird und im-

mer mit den letzten Ergebnissen aktueller Forschung erweitert wird. Außerdem unterstützt es den *VRML*-Standard und seinen Nachfolger *X3D*, was dieser Arbeit sehr entgegenkommt.
Der Funktionsumfang reicht von einfachen Bildverarbeitungsfilter zur Binarisierung, Grauwertkonvertierung und Linienextraktion bis hin zu komplexen Algorithmen zum Tracken von Gegenständen wie dem KLT-Tracker. Die Projektseite ist unter [34] zu finden. Auf ihr stehen für Linux, Mac und Windows das Releasebuild und auch Dailybuilds zum Download bereit.

2.7.1 Aufbau

Die *Instant Vision* besitzt eine Pipelinearchitektur. Mit dem Herzstück einer Actionpipe. In ihr werden *Action* sequentiell nach dem FIFO-Prinzip[14] abgearbeitet. Eine *Action* beschreibt einen Arbeitsschritt der *Instant Vision* und ist als eine Abstrakte Klasse in C++ geschrieben. Von ihr werden alle Algorithmen und Plug-Ins abgeleitet. Jede *Action* besteht aus verschiede-

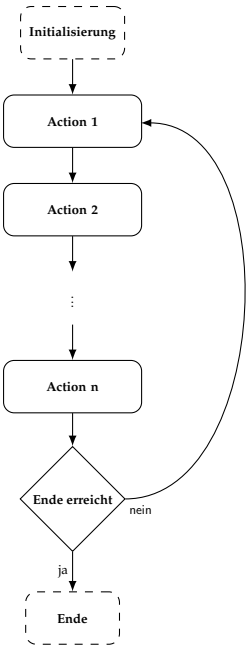

Abbildung 16: Architektur der *Instant Vision*. Quelle: eigene Erstellung

nen Input- und Outputkeys für die Kommunikation zwischen den *Actions*. Diese können darüber hinaus mit Attributen versehen werden, um Parameter anzupassen. Eine *Action* kann nicht direkt mit einer anderen *Action* kommunizieren. Dies geschieht immer über das *Dataset*. In ihr werden alle Daten abgelegt und den *Actions* zur Verfügung gestellt. Eine *Action* hat

[14] *First in first out*

Lese- und Schreibzugriff auf das gesamte *Dataset*. Diese Zugriffe sind zwingend notwendig, da sonst eine *Action* weder Daten zum verarbeiten erhalten kann, noch Ergebnisdaten zur Verfügung stellen kann. Abbildung 16 verdeutlicht die sequentielle Abarbeitung und Kommunikation zwischen einer *Action* und dem *Dataset*.

2.7.2 Action

Dieser Abschnitt soll den internen Aufbau einer *Action* kurz erläutern. Dadurch, dass alle *Actions* von einer abstrakten Klasse abgeleitet werden, ist es sehr komfortabel für *Instant Vision* neue *Actions* anzufertigen. Es werden zwei Methoden geerbt und diese müssen implementiert werden. Zum einen *init()* und zum anderen *apply()*.
Die *init()*-Methode dient zur Initialisierung der *Action*. Ihr wird ein Zeiger auf das *Dataset* übergeben, mit dem sie Daten zur eigenen Vorbereitung lesen und für folgende *Actions* bereitstellen kann. Sie wird beim Öffnen eines Projekts automatisch ausgeführt und soll auch bei Änderungen am Projekt oder Neuerstellung manuell ausgeführt werden.
Die *apply()*-Methode führt den eigentlichen Algorithmus aus. Sie wird beim starten der *Actionpipe* immer wieder ausgeführt. Auch sie erhält einen Zeiger auf das *Dataset*.
Des Weiteren kann eine *Action* andere *Actions* selbstständig erstellen und aufrufen. Zur Kommunikation können zudem eigene *Datasets* angelegt und verwaltet werden.

3 Stand der Technik

In diesem Kapitel sollen Arbeiten vorgestellt werden, die sich ebenfalls mit markerlosem Tracking befassen. Im Kontext dieser Arbeit soll das Hauptaugenmerk auf das Finden der initialen Pose geworfen werden. Das Finden ebendieser lässt sich dabei in zwei große Gruppen unterteilen. Die erste Gruppe sind die *technikgestützten Posenbestimmungen* und die zweite umfasst die *benutzergestützten Posenbestimmungen*. Teilweise ist der Übergang zwischen ihnen fließend und sie gehören genau genommen beiden Gruppen an. Ist dies der Fall, werden sie der Zweiten zugeordnet.

3.1 Technikgestützte Posenbestimmung

Die *technikgestützten Posenbestimmungen* bieten den großen Vorteil, dass sie nicht auf die Angaben eines Nutzers angewiesen sind. Außerdem ist nicht zu vernachlässigen, dass vorausgesetzt wird, dass der Nutzer ein räumliches Vorstellungsvermögen besitzt, überlässt man ihm die Aufgabe. Auch, wenn wie in Kapitel 3.2 angegeben, die Verfahren dem Nutzer viel Arbeit abnehmen, ist es für sie zwingend erforderlich, dass der Nutzer ein räumliches Verständnis besitzt, um das 3D Modell korrekt zu positionieren.

3.1.1 Trainingsbilder

In [35, 36, 37, 38, 39, 40] werden Verfahren angegeben, die anhand von Trainingsbildern die Posen bestimmen können. Dazu wird das System mit einer Menge von Bildern angelernt. Aus diesen Bildern wird Vorwissen gesammelt, sodass das System bereits vor Beginn des Trackings weiß, wonach gesucht wird. In [37] werden beispielsweise 70 Bilder verwendet. Der vorgeschlagene Ansatz arbeitet in zwei Schritten. Erstens wird das Objekt im Bild lokalisiert und zweitens seine Pose bestimmt. Beim ersten Schritt werden dazu co-occurrence Farbhistogramme der Trainingsbilder und dem Kamerabild erstellt und verglichen. Im zweiten Schritt wird anhand eines 3D Modells die Pose geschätzt. Die Autoren geben an, dass sie im schlechtesten Fall eine Abweichung von 20° hatten und im Durchschnitt von 6°. Weiter wird erwähnt, dass bereits 50 Trainingsbilder ausreichend sind. Mehr Trainingsbilder bewirken keine signifikante Verbesserung mehr. In Abbildung 17 wird das Ergebnis der Vergleiche der co-occurrence Farb-

histogramme gezeigt. Im rechten Bild ist die Bewertungsmatrix der orange Reisschachtel dargestellt. Um so dunkler die Matrix an einer Stelle ist, desto besser haben die Trainingsbilder der Schachtel an dieser Stelle mit dem Kamerabild übereingestimmt. Wie in [39] unter *Kapitel 5.2 Matching Rates*

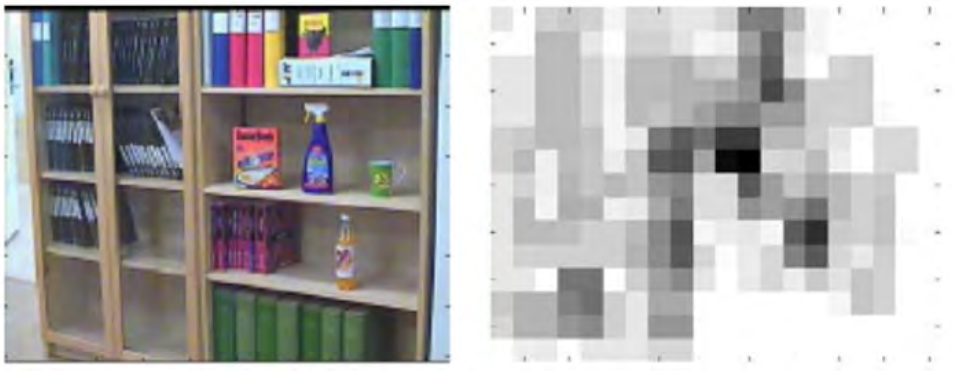

Abbildung 17: Posenbestimmung mit Trainingsbilder nach [37]. Quelle: [37]

angegeben, hängt der Erfolg der Posenbestimmung anhand von Trainingsbildern stark von den Trainingsbildern selbst ab. So führen die Autoren auf, dass ein Testszenarion nicht erfolgreich abgeschlossen werden konnte, weil das Trainingsset ein solches Szenario nicht umfasste. Wie in [41] treffend formuliert, sollten die Trainingsbilder auf die spätere Anwendungsdomäne abgestimmt sein.

3.1.2 Segmentierung

[42] gibt ein Verfahren an, das nicht die komplette Pose mit sechs Freiheitsgraden findet, sondern sich auf die drei Freiheitsgrade der Translation beschränkt. Vielmehr ist dieser Ansatz ein 2D Tracking zur Mensch-Computer-Interaktion [15] und kein 3D Tracking. Die fehlende Dimension kann laut den Autoren durch eine weitere Kamera gewonnen werden.
Die Idee ist es, das Kamerabild möglichst eindeutig zu segmentieren. Dabei wird eine Segmentierung nach Mumford-Shah genutzt[43]. Ziel des Algorithmus ist es, das Bild in insgesamt sechs Interessensbereiche aufzuteilen. Wie in Abbildung 18 (rechts) leicht zu erkennen ist, sind die Bereiche wie folgt aufgeteilt: Rumpf, Kopf, Hand 1, Hand 2, Haare und Hintergrund. Das System arbeitet mit einer Farbdistanz, anhand derer bewertet wird, ob ein Segment als Haut (Hand oder Gesicht) erkannt wird. Dazu wird dem System ein Farbbereich für Hauttöne gegeben. Das größte Segment wird dem Gesicht Z zugeordnet. Daran schließt sich nach oben das Segment X

[15] *Wikipedia (besucht: 27 März 2013):* "Die Mensch-Computer-Interaktion (englisch Human-Computer Interaction, HCI) als Teilgebiet der Informatik beschäftigt sich mit der benutzergerechten Gestaltung von interaktiven Systemen und ihren Mensch-Maschine-Schnittstellen."

für die Haare an. Das größte Segment unter Z wird dem Rumpf W zugeordnet. Segmente links und rechts von Z, die im Farbbereich der Hauttöne liegen, werden der linken und rechten Hand Y_1, Y_2 zugewiesen. Wie in Abbildung 18 (links) zu erkennen ist, sind die Segmente nicht zwangsläufig zusammenhängend. Ziel ist es, W an Z, Y_1 und Y_2 anzuschließen. Dazu wird t_i nach der initialen Segmentierung je nach räumlicher und farblicher Distanz, an W, Z, Y_1 oder Y_2 angefügt. Wurden alle t_i eingegliedert, wird von Z, Y_1 und Y_2 jeweils der Schwerpunkt berechnet (in Abbildung 18 mit roten Quadraten gekennzeichnet). Anhand eines Livebildes können die Schwerpunkte registriert und ausgewertet werden. Laut Autoren ist dieser Algorithmus echtzeitfähig.

Abbildung 18: Posenfindung durch Segmentierung. Quelle: [42]

3.1.3 Analyse durch Synthese

Als letzten Ansatz der *technikgestützten Posenbestimmungen* soll hier das Verfahren der *Analyse durch Synthese* vorgestellt werden. Als interessantes Forschungsfeld hat sich herausgestellt, dass über GPS-Koordinaten und Kompassdaten ein *Grobtracking* durchgeführt und durch *Analyse durch Synthese* die Pose verfeinert wird (*Feintracking*). Abbildung 19 verdeutlicht dieses Vorgehen. Im linken Bild ist das Kamerabild zu sehen und mittig die Pose resultierend aus den beiden Sensoren. Rechts sieht man das Ergebnis nach der Verfeinerung durch *Analyse durch Synthese*.

Abbildung 19: Posenfindung durch GPS, Kompass und *Analyse durch Synthese*. Quelle: [44]

Zwei Arbeiten, die sich mit dieser Herangehensweise beschäftigt haben,

sind [2, 44]. *Analyse durch Synthese* ist ein Ansatz, der ein Kamerabild mit einem synthetisch erstellten Bild vergleicht. In [2, 44] ist das synthetische Bild ein gerendertes texturiertes 3D-Modell. Dazu werden mehrere Bilder gerendert, die rund um die grobe Pose gestreut werden. Durch Optimierungsverfahren, wie beispielsweise einem Partikelfilter, wird das Streuen optimiert und die exakte Pose angenähert, bis ein Fitnesswert zwischen Kamerabild und synthetischem Bild erreicht ist. Bildlich kann man sich dieses Verfahren so vorstellen, dass an der Kamera gewackelt wird, während sie Bilder rendert. Durch eine Vergleichsmethode wird dann das beste Bild als Pose angenommen. Der Schritt des Wackelns, also dem Erstellen verschiedener synthetischer Bilder, kann optimiert werden, um sich möglichst schnell der exakten Pose anzunähern.
Als ein Problem dieses Verfahrens hat sich in [44] herausgestellt, dass das 3D-Modell möglichst exakt und proportional zu dem realen Objekt sein muss. Beispielsweise können nicht modellierte Details das Ergebnis erheblich beeinflussen. In Abbildung 20 (links) wurden nicht alle Fenster des Gebäudes modelliert, wodurch wird eine zuverässige Posenschätzung erschwert wird. Ebenfalls wird dieses Verfahren von einer unpräzisen GPS-Ortung negativ beeinflusst. Da die synthetischen Posen nur um die grobe Pose der Sensoren gestreut wird, kann der Bildvergleich kein passendes Ergebnis finden. Dieses Problem könnte man auf Kosten der Performance ausgleichen, indem die Posen weiter gestreut werden [44]. Als dritte Problematik sei die Abhängigkeit gegenüber GPS nochmals erwähnt. Dadurch ist lediglich ein Outdoor-Tracking möglich und es würde, so wie es in [2, 44] gemacht wird auch nicht im Innenbereich funktionieren.

Abbildung 20: *Analyse durch Synthese*: unvollständiges Modell. Quelle: [44]

3.2 Benutzergestützte Posenbestimmung

Die *benutzergestützten Posenbestimmungen* lassen sich dadurch beschreiben, dass der Benutzer aktiv werden muss. Aktiv bedeutet, dass er beispielsweise das auszurichtende 3D-Modell an Ort und Stelle per Hand schieben muss (Translation) und eventuell auch die Orientierung, also die Rotation

angeben muss.

3.2.1 Bewegen der Kamera

In [45] wird ein Ansatz angegeben, der zum einen ein Trackingverfahren vorschlägt, das sich aus Kombinationen von Sensoren zusammensetzt und zum anderen ein vorgefertigtes CAD[16] Modell zur Feature-Registrierung benutzt.
Die Initialpose und auch die Reinitialisierung werden semi-automatisch gefunden. Insgesamt sind zwei Schritte dazu notwendig. Der erste Schritt ist die Auswertung der Sensordaten des *IMU*[17], um eine grobe erste Orientierung zu erhalten. Der zweite Schritt umfasst die Kalibrierung der Position. Diese wird manuell gemacht, indem der Benutzer die Kamera möglichst exakt an die Stelle des 3D-Modells bewegt.
Wie unter *Conclusion and future Work* angegeben, wollen die Autoren zukünftig auf ein vorbereitetes CAD-Modell verzichten. Außerdem wird darauf hingewiesen, dass die semi-automatische (Re-)Initialisierung für größere Umgebungen nicht ausreichend ist. Abbildung 21 zeigt 4 Screens-

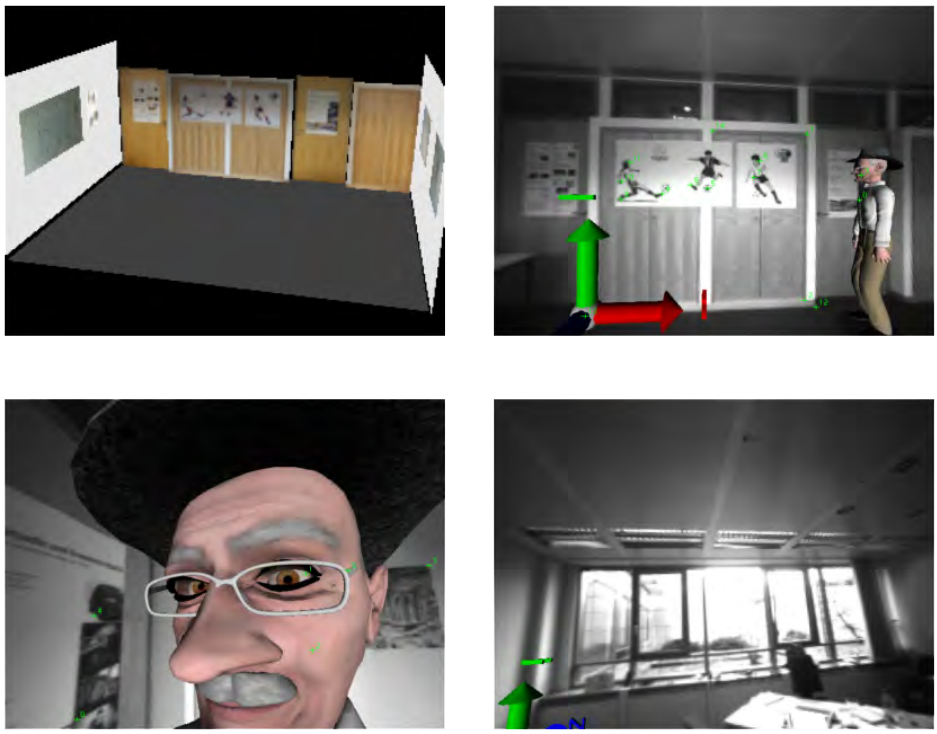

Abbildung 21: Screenshots zu [45]. Quelle: [45]

[16] Computer-Aided Design
[17] Inertial Measurement Unit - Sensor zur Messung von Beschleunigungen und Drehraten

hots dieses Ansatzes. Oben links ist das CAD-Modell, das für die Feature-Registrierung genutzt wird, zu sehen. Die restlichen Bilder geben einen Überblick des Bewegungsradius. Die Wand, die im Bild unten rechts zu sehen ist, ist nicht Teil des Modells, dennoch funktioniert die Augmentierung im Bild links unten.

In [46] wird ein ähnlicher Ansatz verfolgt, was das Finden der Initialpose angeht. Der Nutzer nähert das 3D-Modell dem Kamerabild möglichst genau durch Bewegen der Kamera an. Ihm stehen die sechs Freiheitsgrade der Rotation und Translation zur Verfügung. Aufgrund der Problematik der Initialisierung hat sich der Autor dieser Arbeit schon früh wieder von diesem Schritt gelöst und sich seiner eigentlichen Problemstellung gewidmet. Er hat den Ansatz der *Analyse durch Synthese* verfolgt. Die Idee dieser Arbeit war es, das Objekt möglichst photorealistisch zu rendern, um es ohne zusätzlichen Bearbeitungsschritt mit dem Kamerabild vergleichen zu können. Nennenswert ist an dieser Stelle, dass dazu eine HDR[18]-Kamera mit Fischaugenobjektiv verwendet wurde, um die Beleuchtungssituation möglichst genau nachbilden zu können. Abbildung 22 zeigt den Aufbau des zu erfassenden Objekts und die technischen Hilfsmittel - einer Farbvideokamera und der HDR-Kamera. Wie auch schon in Kaptitel 3.1.3 be-

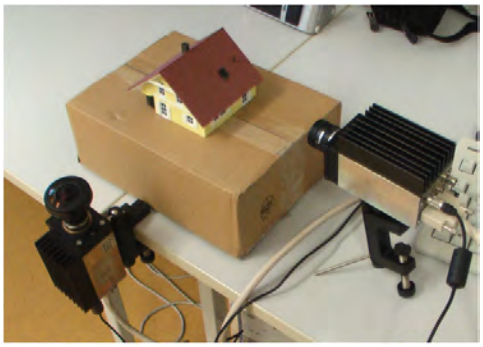

Abbildung 22: Screenshots zu [46]. Quelle: [46]

schrieben, hängt der Erfolg von der Qualität und Vollständigkeit des Modells ab. Des Weiteren lieferte die HDR-Kamera ein hohes Grundrauschen und eine geringe Auflösung, wodurch die Punktdetektion mit SIFT[47] verschlechtert wurde.

Als letzte Arbeit dieser Art sei [48] erwähnt. Screenshots der laufenden Augmentierung ist in Abbildung 23 zu sehen. Auch hier wird die initiale Kamerapose durch das Annähern des Kamerabildes durch Bewegen der Kamera an das 3D-Modell angenähert. Dieser Schritt soll laut den Autoren möglichst genau geschehen, um ein Konvergieren gegen ein lokales

[18] High Dynamic Range

Extremum zu vermeiden. Daraus lässt sich schließen, dass auch Ansätze, die sich auf eine *benutzergestützte Posenbestimmung* verlassen, nicht das Problem des lokalen Extremum gelöst haben.

Abbildung 23: Screenshots zu [48]. Quelle: [48]

3.2.2 Bewegen des 3D-Modells

Das nächste Verfahren dieser Gruppe sei eine kommerzielle Lösung der Firma *Metaio* [49]. Es besteht - wie schon [45] - aus einer Kombination aus verschiedenen Sensoren und einer Benutzerinteraktion und eignet sich hauptsächlich für Outdoor-Tracking. Zum einen wird die Position über *GPS* bestimmt und die Orientierung anhand einer Gleichfeldmessung (Kompass) gewonnen. Beide Sensoren sind in nahezu allen mobilen Geräten mittlerweile zu finden. Allerdings ist durch die Abhängigkeit gegenüber einem GPS-Signal der In-door-Einsatz nicht erfolgversprechend. Zudem bietet GPS eine Ortungsgenauigkeit von nur wenigen Metern, weswegen sich eine manuelle Feinjustierung durch den Nuter zwangläufig anschließt. Dabei bewegt der Benutzer nicht die Kamera, sondern er verschiebt das 3D-Modell auf das reale Objekt. Es reicht meistens schon aus, die Pose auf wenige Dezimeter genau anzugeben, bis sie einrastet und sogar Zentimeter genau sitzt. Die Maßeinheiten sind in Bezug auf die vorgestellte Demo auf der *InsideAR 2012* und können abweichen, da von *Metaio* kein wissenschaftliches Paper über dieses Verfahren veröffentlich wurde. Abbildung 24 zeigt

Abbildung 24: Screenshots zu Metaios Outdoor Tracking. Quelle: [50]

im linken Bild die grobe Pose anhand GPS und dem Kompass. Im rechten

Bild wird das Einrasten von der groben Pose (rot) zu der angepassten Pose (grün) gezeigt, nachdem der Benutzer das Linienmodell angepasst hat.

3.2.3 Eingabe von 3D-2D-Korrespondenzen

In [40] wird ein etwas anderes Verfahren verfolgt. Hier beeinflusst nicht der Nutzer die Pose der Kamera, sondern er gibt manuell 3D-2D- Korrespondenzen an. Aus diesen Korrespondenzen kann, wie in Kapitel 2.4 angegeben, eine exakte Pose berechnet werden. Abbildung 25 zeigt links das

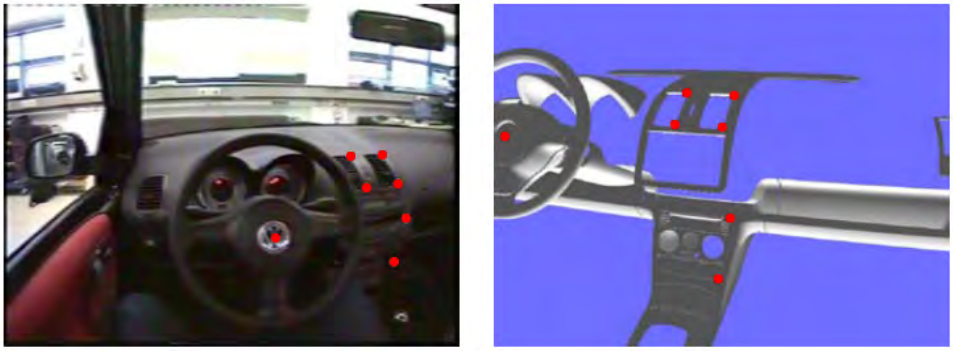

Abbildung 25: Screenshots zu [40]. Quelle: [40]

Kamerabild und rechts das 3D-Modell. Die roten Punkte zeigen die vom Benutzer erstellten Korrespondenzen zwischen den Bildern. Wie in [40] angegeben, lassen sich durch dieses Verfahren stabile und exakte Ergebnis erzielen.

3.3 Genetischer Algorithmus

Wie schon im Kapitel 2.5.3 erläutert sind *genetische Algorithmen* stark an die Biologie angelehnt. Sie verwenden den gleichen Aufbau und die gleichen Methoden, um eine Lösung zu verbessern. Das grundsätzliche Verfahren eines *genetischen Algorithmus* hat sich über die Jahre, seit der Einführung in den 1960er Jahren, nicht erheblich geändert.

3.3.1 Verteilte Evolution

Verteilte Evolution (distributed Evolution) geht zurück auf das biologische Prinzip der Insel Evolution. Die Idee ist es, verschiedene Populationen getrennt voneinander entwickeln zu lassen. Wie es bereits schon in Kapitel 2.5.3.9 beschrieben wurde. Verschiedene Ansäzte lassen sich auf die Fragestellung der Migration finden. Abbildung 26 verdeutlicht das grundsätzliche Vorgehen einer Insel Evolution mit vier Inseln. Nach jeder Epoche findet ein Austausch von n Individuen statt. Wie eine Epoche definiert ist und wie der Austausch stattfinden kann, wird nachfolgend erläutert.

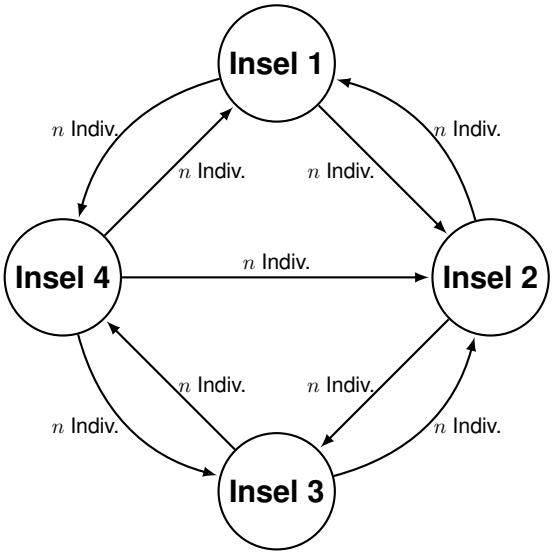

Abbildung 26: Darstellung der Insel Evolution während einer Migration. Quelle: eigene Erstellung

Bereits der erste *genetische Algorithmus*, angegeben in [32], verwendete eine zufällige Auswahl der Migranten. Die zufällige Auswahl folgt keiner wahrscheinlichkeitstheoretischen Verteilung, sondern ist absolut willkürlich. Dadurch soll eine Katastrophe oder auch ein ökologischer Wechsel auf der Insel simuliert werden. Wie sich später herausstellte, hat sich dieses Verfahren als das Beste erwiesen und etablierte sich. Das Auftreten einer Migration wurde hier nach einer fixen Anzahl an Generationen durchgeführt. Diese Anzahl wird an dieser Stelle als Epoche eingeführt. Wie in [51] gezeigt, ist eine variable Epochenlänge besser. Um dies umzusetzen, definieren die Autoren das Ende einer Epoche zu dem Zeitpunkt, wenn sich das Individuum mit der besten Fitness seit 25 Generationen nicht mehr verbessert hat.
Weiter zeigen die Autoren, dass eine willkürliche Wahl der Migranten effektiver ist und eine fitnessproportionale wesentlich häufiger in einem lokalen Extrema (Kapitel 2.5) hängen bleibt.

In [52] wird ein Verfahren zur Migration angegeben, das nach jeder Generation jede Insel die besten Individuen zu den anderen Inseln abwandern lässt und dort die schlechtesten durch die Gesandten ersetzt werden. [53, 54] geben eine fixe Epochenlänge an und die Migranten werden abhängig zu ihrer Fitness selektiert. Die Insel, die diese Individuen erhält, ersetzt nicht die schlechtesten wie in [52], sondern wählt zufällig Individuen aus. Bedingung ist allerdings, dass ihre Fitness schlechter ist, als die der Ge-

sandten.
[55] erweitert die Arbeit von [54], indem die Migranten zufällig ausgewählt werden.

In [33] wird angegeben, dass für Probleme bei denen eine Vergrößerung der Population keine besseren Ergebnisse erzeugt, die Verteilung auf verschiedene Inseln eine Verbesserung bezweckt. Der Autor gibt an, dass bei der Verwendung von M 'Inseln', die Population N_{total} gleichmäßig aufzuteilen und nicht, wie man vermuten könnte, jede Insel mit N_{total} initialisiert. Daraus ergibt sich Gleichung 3.3.1.

$$N_{island} = \frac{N_{total}}{M} \tag{3.3.1}$$

Dabei gibt N_{total} die Anzahl der Individuen auf einer Insel an. [33] hat die Insel Evolution anhand verschiedener mathematischer Funktionen getestet. In Abbildung 27 sieht man den Vergleich eines *genetischen Algorithmus* mit (links) und ohne Migration. Wie man leicht erkennen kann, ist das Ergebnis mit Migration bei allen Populationsgrößen besser. Ebenfalls gut zu sehen ist, dass es ohne Migration spätestens nach der 175.000 Generation keine Verbesserung mehr gibt. Mit Migration hingegen verbessert sich das Ergebnis auch noch nach 375.000 Generationen. Dies ist ein Indiz dafür, dass man mit Migration dem Problem des lokalen Extrema besser entgegenwirken kann.

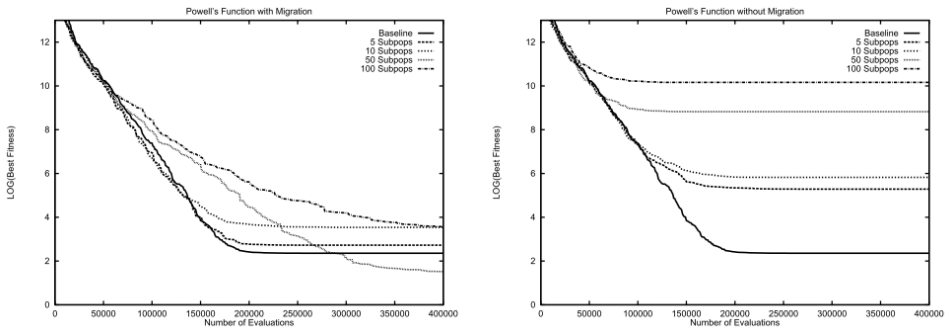

Abbildung 27: Vergleich *genetischer Algorithmus* mit und ohne Migration. Quelle: [33]

3.3.2 GPGPU

Neben der Insel Evolution als systematische Änderung gab es in den letzten Jahren Anstrengungen, *genetische Algorithmen* auf Grafikkarten zu parallelisieren. [56] und [57] bieten dazu GPGPU[19] Lösungen an. Wie schon

[19] *Wikipedia (besucht: 27 März 2013):*"General Purpose Computation on Graphics Processing Unit (kurz GPGPU, vom Englischen für Allzweck-Berechnung auf Grafikprozessorein-

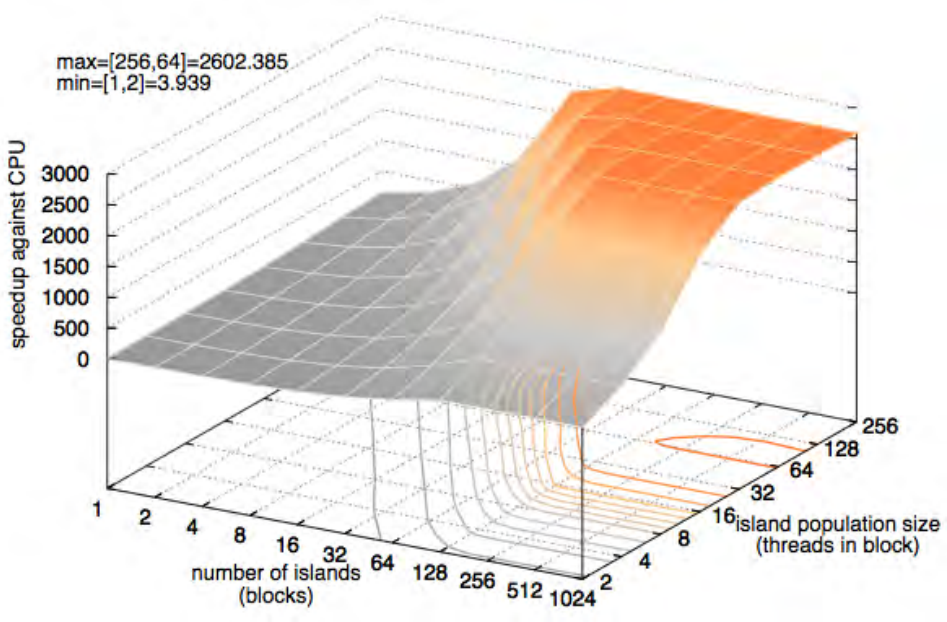

Abbildung 28: Vergleich *genetischer Algorithmus* auf CPU und GPU. Quelle: [56]

erwähnt, kann ein *genetischer Algorithmus* auf mehrere CPUs und sogar auch auf separate Computer verteilt werden. GPGPU Lösungen sind im Vergleich jedoch wesentlich schneller. In Abbildung 28 ist ein Vergleich eines *genetischen Algorithmus* zu sehen, der den Speedup zwischen einer CPU und GPU Implementierung darstellt. Nennenswert ist an dieser Stelle, dass die Testgrafikkarte bereits zwei Jahre älter war, als die CPU. Trotzdem ist sie bis zu 2.600 mal schneller als die CPU. In anderen Worten kann die GPU die Aufgabe, die zuvor Stunden gedauert hat, in Sekunden lösen.
Weiter geben die Autoren an, dass die Ergebnisse der GPU qualitativ ähnlich mit denen der CPU sind. Allerdings sei erwähnt, dass eine GPGPU-Implementierung nicht trivial ist und ein nicht zu unterschätzendes Verständnis der unterliegenden Hardware erfordert.

heit(en)) bezeichnet die Verwendung eines Grafikprozessors für Berechnungen über seinen ursprünglichen Aufgabenbereich hinaus."

4 Konzept

In diesem Kapitel soll das Konzept erläutert werden, das im Zuge dieser Bachelorarbeit entstanden ist, um das Posenfindungsproblem zu lösen. Es wurde sich zum Ziel gesetzt, mit einem möglichst einfachen 3D Modell auszukommen, das weder texturiert, beleuchtet, noch eine hohe Detailtiefe besitzen muss. Die einzigen Informationen, die der Algorithmus benötigt, sind zwei Mengen mit 2D und 3D Feature. Wie diese aus einem Kamerabild für 2D Feature und für 3D Feature aus einer Bildsequenz oder 3D Modell gewonnen werden können, wurde in den Kapiteln 2.1 und 2.2 vorgestellt.

4.1 Idee

Die Idee ist es, zwischen diesen Mengen 3D-2D Korrespondenzen herzustellen. Wie im Kapitel 2.3 erläutert, ist es möglich aus sechs solcher Korrespondenzen eine Pose zu schätzen. In dieser Arbeit wurde die Möglichkeit der linearen Posenschätzung genutzt. Wie beschrieben, benötigt die iterative Schätzung eine Startpose, die sie Schritt für Schritt verbessert. Im Gegensatz dazu steht die lineare Schätzung. Ihr reichen als Übergabeparameter lediglich sechs Korrespondenzen.
In Abbildung 29 wird dieses Vorgehen illustriert. Im linken Koordinaten-

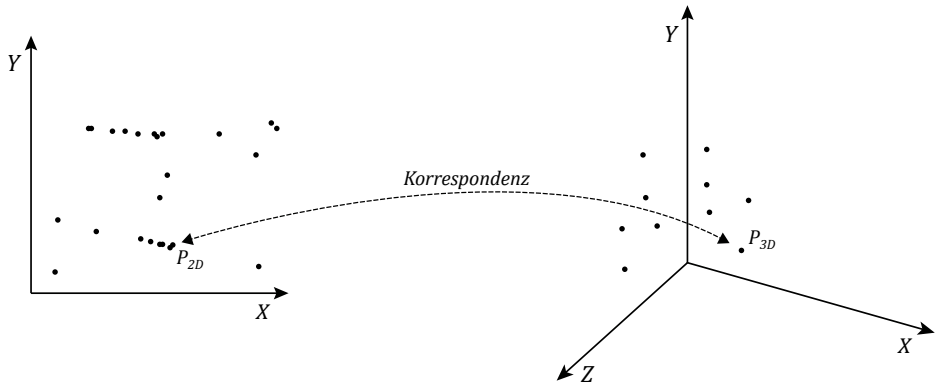

Abbildung 29: Eine Korrespondenz zwischen 2D Container (links) und 3D Container (rechts). Quelle: eigene Erstellung

system sind die 2D Feature aufgetragen und im rechten die 3D Feature. Wie man leicht erkennen kann, schlägt ein 2D Featuredetektor nicht nur dort an, wo man auch Feature im 3D Modell vorfindet. In der Regel befinden sich noch andere Objekte oder ein texturierter Hintergrund im Kamerabild. Außerdem finden Punktdetektoren verstärkt in verrauschten Bildern Feature, die nicht von der Umgebung stammen, sondern von dem Kamerarauschen. Nähers dazu kann Kapitel 4.3.1 entnommen werden. In Abbildung 30 ist eine Möglichkeit illustriert, wie die beiden Mengen durch sechs Korrespondenzen verbunden werden können. Die resultierende Pose wäre in diesem Beispiel sehr ungenau. Verwendet man keinen Schätzer, wie *M-Schätzer* oder *RANSAC*, verursacht bereits eine falsche Korrespondenz eine inkorrekte Pose. Schätzer analysieren die gegebenen Korrespondenzen und versuchen Ausreißer, die nicht korrekt sind, für weitere Berechnungen zu entfernen. Weitere Informationen dazu können [3] entnommen werden.

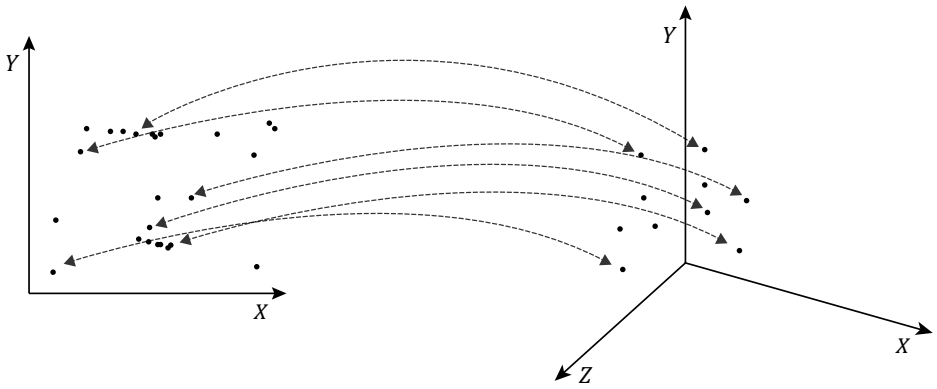

Abbildung 30: Sechs eindeutige Korrespondenzen zur Bestimmung einer Pose. Quelle: eigene Erstellung

Nachdem aus den sechs Korrespondenzen eine Pose (K_e) geschätzt wurde, muss diese bewertet werden. Da zu diesem Zeitpunkt P_{3D} und K_i bekannt sind und als richtig angenommen werden können, kann mit Formel 2.3.15 ein 3D-Feature in 2D-Bildkoordinaten übertragen werden und K_e bewertet werden. Zur Bewertung wurde folgender Algorithmus entwickelt:

- *1. Schritt:* Wähle eine noch nicht bewertete 3D-2D-Korrespondenz.
- *2. Schritt:* Berechne P'_{3D} mit Formel 2.3.15.
- *3. Schritt:* Berechne den *Euklid'schen Abstand* zwischen P'_{3D} und seinem korrespondierendem 2D-Feature.
- *4. Schritt:* Setze die Distanz der Korrespondenz auf den zuvor berechneten Wert.

- *5. Schritt:* Wurden alle Korrespondenzen bewertet, gehe zu *6. Schritt*, sonst zu *1. Schritt*.
- *6. Schritt:* Summiere alle Distanzen auf und weise es Ψ als Gesamtdistanz von K_e zu.

Wichtig ist bei diesem Verfahren zu beachten, auch für spätere Implementierungen, dass die Distanz nicht als Fitness missverstanden werden darf. Sie ist eine Metrik, die angibt, wie schlecht K_e ist und nicht wie gut. Je kleiner Ψ eines K_e, desto besser. Diese Problematik wird in Kapitel 4.4.5 noch einmal aufgegriffen.
Wurde ein $\Psi \leq 6$ gefunden, kann das zugehörige K_e als gültige Pose angenommen werden. Wie sich im Laufe der Entwicklung gezeigt hat, liefert ein Abbruchkriterium $\Psi \leq 10$ bereits die korrekte Pose.

4.2 Suchraum der Posenbestimmung

Da ein möglichst allgemeingültiger Ansatz entwickelt werden soll, besitzt man keine Informationen über die Güte der 2D-Feature. Man muss davon ausgehen, dass alle 2D-Feature relevant sein können. Bei den 3D-Feature verhält es sich ähnlich. Da wir nicht wissen, wie das Objekt im Kamerabild positioniert ist, müssen auch alle 3D-Feature in Betracht gezogen werden. Die Selbstverdeckung eines Objekts wird somit nicht beachtet. Bei diesen Voraussetzungen wurde für die vorliegende Arbeit eine zufallsgesteuerte Korrespondenzfindung genutzt. Das bedeutet, dass sechs Korrespondenzen für eine Posenberechnung zufällig gewählt werden. Dass sich daraus ein großer Suchraum ableiten lässt, zeigt dieses Kapitel.

4.2.1 Urnenmodel

Das angegebene Beispiel aus Abbildung 30 besitzt insgesamt zehn 3D- Feature und 24 2D-Feature. Wie viele Möglichkeiten es gibt sechs Feature jeweils aus der 3D-Menge und 2D-Menge auszuwählen, lässt sich anhand eines klassischem Urnenmodels beschreiben. Vereinfachend wird von nun an für die Menge der 3D-Feature B und für 2D-Feature C genutzt. B besteht in diesem Beispiel aus insgesamt fünf Elementen und lässt sich folgendermaßen notieren:

$$B = \{b_1, b_2, b_3, b_4, b_5\}.$$

C wird auch mit fünf Elementen angenommen:

$$C = \{c_1, c_2, c_3, c_4, c_5\}.$$

Zur besseren Veranschaulichung kann die Auswahl der Korrespondenzen als das Ziehen einer Untermenge von B und C beschrieben werden. Das

Ergebnis einer Auswahl ist eine Teilmenge der jeweiligen Übermenge. Es gilt

$$B_i \subset B, \tag{4.2.1}$$

wobei B_i die i-te Teilmenge beschreibt. Analog gilt das für C. Wie viele Teilmengen möglich sind, gilt es nun herauszufinden. In unserem kleinen Beispiel sollen lediglich drei Elemente aus B und C ausgewählt werden. Übertragen würde dies drei Korrespondenzen bedeuten.
Es gibt insgesamt

$$n^k \tag{4.2.2}$$

Variationen. Also $5^3 = 125$ Möglichkeiten drei (k) aus 5 (n) zu ziehen. Es sind also Wiederholungen möglich und die Reihenfolge spielt eine Rolle. Es sind unter anderem folgende Teilmengen möglich:

$$B = \{(b_1, b_1, b_1), (b_1, b_2, b_3), (b_1, b_3, b_2), ...\}.$$

$$C = \{(c_1, c_1, c_1), (c_1, c_2, c_3), (c_1, c_3, c_2), ...\}.$$

Wir wählen nun jeweils ein Element, um daraus drei Korrespondenzen zu erhalten. Beispielsweise B_1 und C_2.

$$B_2 = (b_1, b_2, b_3) \text{ und } C_1 = (c_1, c_1, c_1).$$

Nun wird jeweils aus den i-ten Elementen eine Korrespondenz hergestellt. Wir erhalten folgendes Ergebnis:

1. $b_1 \Longleftrightarrow c_1$
2. $b_2 \Longleftrightarrow c_1$
3. $b_3 \Longleftrightarrow c_1$

Vereinfachend kann auch

$$B_1 \Longleftrightarrow C_2$$

geschrieben werden. Wie man erkennt, wurde das 2D-Feature c_1 in allen Korrespondenzen genutzt. Dies ist auch richtig. Denn mehrere 3D-Feature können auf einen 2D-Bildpunkt abgebildet werden. Umgekehrt geht dies nicht. Da in unserem Fall ein 3D-Feature nicht auf mehrere 2D-Feature abgebildet werden darf, muss auch die Formel zur Berechnung der Variationen angepasst werden. Weiterhin ist die Reihenfolge bedeutend, allerdings muss für die Anzahl der möglichen Variationen der 3D-Feature die Wiederholungen herausgenommen $\{(1, 1, 1), (1, 2, 2), ..., (5, 5, 5)\}$ werden. Die Formel für das Urnenmodell mit Reihenfolge, aber ohne Wiederholungen lautet:

$$\frac{n!}{(n-k)!}. \tag{4.2.3}$$

Die Reihenfolge muss beachtet werden, damit auch Korrespondenzen wie $B_1 \Longleftrightarrow C_2$ und $B_1 \Longleftrightarrow C_3$ als unterschiedliche Korrespondenzen mit in

die Rechnung aufgenommen werden. Nach Formel 4.2.3 ergeben sich 60 Variationen, drei Elemente aus B zu ziehen. Wie man sieht, befinden sich in der Menge B keine Wiederholungen mehr. Das Element $\{b_1, b_1, b_1\}$ ist nun beispielsweise nicht mehr möglich.

$$B = \{(c_2, c_1, c_3), (b_1, b_2, b_3), (b_1, b_3, b_2), (b_1, b_4, b_5), (b_4, b_1, b_5), ...\}$$

$$C = \{(c_1, c_1, c_1), (c_2, c_1, c_3), (c_1, c_2, c_3), (c_1, c_3, c_2), (c_1, c_4, c_5), ...\}$$

Aber auch bei dieser Berechnung kommt es zu einem Problem. Bildet man nun die Korrespondenzen $B_4 \iff C_3$ und $B_5 \iff C_2$, erkennt man in ausgeschriebener Form schnell, dass zwei Mal exakt die gleichen Korrespondenzen gebildet wurden. Diese Dopplungen gilt es in der Berechnung ebenfalls zu vermeiden. Gelöst wird dieses Problem, indem die Reihenfolge für die 2D-Feature keine Rolle spielt. Somit würden die Teilmengen C_2 und C_3 nicht mehr zusammen in C vorkommen. Ausgeschrieben ergibt sich:

$$\begin{aligned} &1.\ b_1 \iff c_1 \\ &2.\ b_4 \iff c_2 \\ &3.\ b_5 \iff c_3 \end{aligned}$$

und

$$\begin{aligned} &1.\ b_4 \iff c_2 \\ &2.\ b_1 \iff c_1 \\ &3.\ b_5 \iff c_3. \end{aligned}$$

In beiden Fällen handelt sich um die gleichen Korrespondenzen: Um eine Korrespondenz zwischen b_1 und c_1, b_4 und c_2, b_5 und c_3. Man erhält also zweimal die gleiche Pose.
Beachtet man bei einem Urnenmodell nicht die Reihenfolge, spricht man nicht mehr von Variationen sondern von Kombinationen. Die Berechnung der Kombinationen der 2D-Feature ist angegeben in Formel 4.2.4

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!} \tag{4.2.4}$$

und ergibt eingesetzt

$$\binom{5}{3} = \frac{5!}{3! \cdot (2)!} = 10\ .$$

Zusammenfassend werden die Variationen der 3D-Feature mit Formel 4.2.3 berechnet und die Kombinationen der 2D-Feature mit Formel 4.2.4.

4.2.2 Kartesisches Produkt

Wie groß nun der gesamte Suchraum ist, lässt sich mit dem kartesischen Produkt beschreiben. Notiert wird es mit

$$B \times C$$

und ist die Menge aller möglicher geordneter Paare (B_i, C_i), wobei B_i ein Element aus B und C_i ein Element aus C ist. Grafisch darstellen lässt sich ein kartesischen Produkt zweier Mengen anhand einer Tabelle:

	C_1	C_2	C_3	$\cdots$	C_{10}
B_1	$B_1 \Leftrightarrow C_1$	$B_1 \Leftrightarrow C_2$	$B_1 \Leftrightarrow C_3$		$B_1 \Leftrightarrow C_{10}$
B_2	$B_2 \Leftrightarrow C_1$	$B_2 \Leftrightarrow C_2$	$B_2 \Leftrightarrow C_3$		$B_2 \Leftrightarrow C_{10}$
B_3	$B_3 \Leftrightarrow C_1$	$B_3 \Leftrightarrow C_2$	$B_3 \Leftrightarrow C_3$		$B_3 \Leftrightarrow C_{10}$
$\vdots$			$\vdots$	$\ddots$	
B_{60}	$B_{60} \Leftrightarrow C_1$	$B_{60} \Leftrightarrow C_2$	$B_{60} \Leftrightarrow C_3$		$B_{60} \Leftrightarrow C_{10}$

Es wird jedes Element der Mengen miteinander kombiniert und man erhält insgesamt $60 \times 10 = 600$ Möglichkeiten. Dies ist die Anzahl auf wie viele verschiedene Arten man drei Korrespondenzen zwischen beiden Mengen herstellen kann.

4.2.3 Geschlossene Formel

In den beiden vorherigen Kapiteln wurde erläutert, wie man den Suchraum für das gegebene Beispiel berechnen kann. In diesem Abschnitt soll dazu eine geschlossene Formel angegeben werden.
Die Anzahl der möglichen Ziehungen mit Wiederholung unter Beachtung der Reihenfolge für 3D-Feature, lässt sich mit Formel 4.2.3 berechnen. Die möglichen Ziehungen ohne Wiederholung und beachten der Reihenfolge für 2D-Feature lässt sich mit Formel 4.2.4 berechnen. Die Ergebnisse werden aufgrund des Kapitels 4.2.2 miteinander multipliziert.
Als geschlossene Formel ergibt sich daraus:

$$\gamma_{(F_{3D},F_{2D})} = |F_{3D}|^k \cdot \frac{|F_{2D}|!}{k! \cdot (|F_{2D}| - k)!}, \tag{4.2.5}$$

wobei k die Anzahl der Korrespondenzen angibt, F_{3D} die 3D-Feature Menge, F_{2D} die 2D-Feature Menge und $|F_{xD}|$ die Anzahl der Elemente in einer Menge.

4.3 Merkmalfindung

Jetzt, wo das grundlegende Verfahren und die Berechnung des Suchraums bekannt sind, soll geklärt werden, wie die 2D-und 3D-Feature für diese

Arbeit detektiert werden. Wie in Kapitel 2.1 und 2.2 beschrieben, gibt es mehrere Möglichkeiten diese Feature zu finden. Die ausgewählten Verfahren wurden aufgrund von Empfehlungen aus der Literatur oder empirisch festgelegt.

4.3.1 2D

In [3] wurden unter anderem verschiedene 2D-Feature Detektoren evaluiert und bewertet. Als besonders gut bezüglich der Bewertungskriterien, wie in Kapitel 2.1 einleitend genannt, hat sich der *FAST Punktdetektor* erwiesen. Ein weiterer Vorteil ist eine bereits vorhandene Implementierung in *Instant Vision*, wodurch das Hauptaugenmerk auf die eigentliche Aufgabenstellung gelegt werden konnte.
Die *Instant Vision* Action *FastFeatureDetectAction* verlangt als Eingabe ein Farbbild. Als Ausgabe erhält man eine Menge 2D-Koordinaten, die jeweils einen gefunden Eckpunkt beschreiben. Diese Menge wird als Eingabe für den entwickelten Algorithmus verwendet. Weitere Informationen der *FAST*-Implementierung der *Instant Vision* können dem Kapitel 5.1 entnommen werden.
Wie in Kapitel 4.1 beschrieben, werden 2D-Feature unter Verwendung eines Eckendetektors auch durch das Bildrauschen[20] erzeugt. Um solche Feature zu vermeiden, wird in der Literatur empfohlen, einen Weichzeichner wie dem *Gaußschen Weichzeichner*, zu verwenden. In Abbildung 31 sieht man die Anwendung eines *FAST Punktdetektor* auf einem verrauschten und weich gezeichnetem Bild. Wie man gut erkennen kann, werden keine falschen Feature mehr gefunden. Man muss aber beachten, dass das komplette Bild weich gezeichnet wird und somit auch Informationen im interessanten Bereich verloren gehen können. Wie solch ein Filter angewandt werden sollte, hängt stark von dem Rauschverhalten der Kamera und den Lichtverhältnissen ab und muss dementsprechend angepasst werden. Da die Aufgabe dieser Arbeit nicht das Finden möglichst guter 2D-Feature war, wurde das Testszenario so gewählt, dass das Bildrauschen möglichst gering ist und kein weiterern Verarbeitungsschritt notwendig sind.

4.3.2 3D

Zur Findung der 3D-Feature wurde in dieser Arbeit die Eigenschaft des *VRML* Formats genutzt. Wie in Kapitel 2.2.1 genannt, gibt es 3D-Datentypen aus denen direkt Feature, ohne weiteren Bearbeitungsschritt, ausgelesen werden können. *VRML* ist solch ein Datentyp.
In Listing 14 ist ein Auszug einer *VRML* Datei angegeben. Sie beschreibt

[20] *Wikipedia (besucht: 14. April 2013):* "Bildrauschen bezeichnet die Verschlechterung eines digitalen bzw. elektronisch aufgenommenen Bildes durch Störungen, die keinen Bezug zum eigentlichen Bildinhalt, dem Bildsignal, haben."

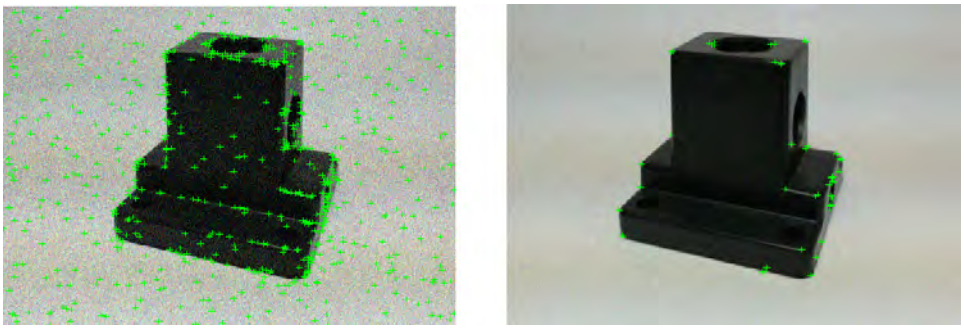

Abbildung 31: Featurevergleich zwischen verrauschtem (links) und weich gezeichnetem Bild (rechts). Der Gaußsche Weichzeichner benutzte einen Radius von 0,6 Pixel und der FAST Punktdetektor arbeitet mit den selben Parametern. Quelle: eigene Erstellung

das 3D-Modell, welches in Abbildung 32 zu sehen ist. Für die Darstellung des *VRML* Formats wurde der *Instant Player*[21] genutzt.

```
1  #VRML V2.0 utf8
2  ...
3    geometry IndexedLineSet {
4      coord Coordinate {
5        point [
6          -2.5 -2.2  2.5,      # point 0, left and close
7          -2.5 -2.2 -2.5,      # point 1, left and back
8           ...
9        ]  # end point
10     } # end coord
11     ...
12   } # end geometry
13 } # end shape
```

Listing 2: Auszug der *VRML* Datei des Lampenständers

In Zeile 1 wird der Datentyp angegeben. Das ist ein großer Vorteil des Formats. Er kann mit einem einfachen Texteditor editiert werden und benötigt kein aufwendiges 3D-Modellierungswerkzeug, wie es andere Formate bräuchten. Zeile 3 gibt an, dass nun die Beschreibung einer Geometrie anhand von Linien erfolgt. Zeile 4 leitet die Angabe der Koordinaten ein und Zeile 5, dass diese Koordinaten als Punkte zu interpretieren sind. Die Zeilen 6 und 7 geben zwei solcher 3D-Punkte an. Diese gilt es aus der Datei zu extrahieren. Da wir auf dem Kamerabild einen Punktdetektor verwenden, der vornehmlich Ecken detektiert, können diese 3D-Punkte verwendet werden. Sie beschreiben die Ecken des zu erfassenden Objekts und stellen somit gefundene 2D-Feature im Kamerabild gewissermaßen im dreidimensionalen Raum dar. Zur Extraktion der 3D-Punkte aus der *VRML* Datei

[21] Der *Instant Player* ist wie *Instant Vision* Teil der *Instant Reality* und wird für die Darstellung des *VRML* Formats genutzt.

wurde ein Parser genutzt, der lediglich die Punkte (bspw. Zeile 6) durch Kommata getrennt ausgibt. Diese Ausgabe dient als zweite Eingabe des entwickelten Algorithmus.

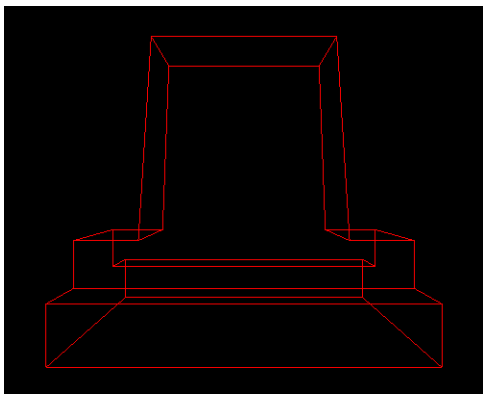

Abbildung 32: 3D Modell unter Verwendung des *VRML* Formats. Quelle: eigene Erstellung

4.4 Genetischer Algorithmus

Tests haben ergeben, dass der Algorithmus, wie in 4.1 angegeben, 2,66 ms benötigt. Selbst mit einem einfachen Beispiel, wie in Kapitel 4.2.1 angegeben, mit jeweils fünf 3D-, 2D-Feature und drei Korrespondenzen, erhalten wir 600 Lösungen. Dies bedeutet, dass die Brute Force Lösung 1,596 Sek. benötigen würde. Ein realistisches Szenario sind beispielsweise 24 3D, 26 2D-Feature und 6 Korrespondenzen. Nach Formel 4.2.5 ergeben sich daraus 22.311.400.000.000 mögliche Lösungen, was eine Laufzeit im schlechtesten Fall von mehreren Jahren bedeutet.
Es ist ganz offensichtlich, dass das Durchlaufen des Suchraums optimiert werden muss. Wie in Kapitel 2.5.3 beschrieben, soll dafür ein genetischer Algorithmus verwendet werden.

4.4.1 Kodierung

Die Kodierung im Kontext eines genetischen Algorithmus beschreibt die Definition eines Individuums. In unserem Fall besteht ein Individuum aus sechs 3D-2D-Korrespondenzen, auf die der Algorithmus aus 4.1 angewandt werden kann und somit eine Lösung beschreibt.

4.4.2 Initialisierung

Die Wahl der 3D- und 2D-Feature sowie der Aufbau einer Population geschieht im Schritt der Initialisierung. Es werden zunächst zufällige 3D-Feature ausgewählt. Die Anzahl ergibt sich aus der Menge der Korrespon-

denzen. Da zur Posenberechnung mindestens sechs Korrespondenzen notwendig sind, müssen auch mindestens sechs 3D-Feature ausgewählt werden. Im nächsten Schritt wird für jeden der sechs 3D-Feature ein 2D-Feature zufällig gewählt. Da ein 3D-Feature mit mehreren 2D-Feature korrespondieren kann, muss nicht darauf geachtet werden, ob ein 2D-Feature bereits Teil einer Korrespondenz ist. Dieses Verfahren wird wiederholt, bis die Population aus der gewünschten Anzahl Individuen besteht. Als weiteren Schritt, was aber nicht unbedingt erforderlich ist, könnte man überprüfen, ob ein Individuum mehrfach vorkommt. Also das Individuum x exakt identisch kodiert ist wie Individuum y. Es hat sich aber gezeigt, dass die Wahrscheinlichkeit für diesen Fall extrem gering ist. Man muss sich an dieser Stelle vor Augen halten, dass es mindestens mehrere Milliarden Möglichkeiten gibt, solch ein Individuum zu kodieren.
Festzuhalten ist, dass die Initialisierung komplett zufällig geschieht, was damit begründet wird, dass wir keinerlei Informationen über das zu erfassende Objekt haben.

4.4.3 Fitnessfunktion

Das Herzstück eines jeden *genetischen Algorithmus* ist die Fitnessfunktion. Sie bewertet jedes Individuum und weist ihm eine Fitness zu, also wie gut oder schlecht es ist. Die geschieht nicht relativ, sondern absolut. In Relation werden sie erst durch das Wahrscheinlichkeitsmaß gebracht (Kapitel 4.4.4). Die Berechnung der Fitness geschieht nicht direkt anhand der Werte eines jeden Individuums. Zunächst wird eine Pose mit einer linearen Methode geschätzt (siehe Kapitel 2.4.1). Anhand dieser Pose kann jedes 3D-Feature einer jeden Korrespondenz eines Individuums in 2D-Bildkoordinaten transliert und projeziert werden, wodurch eine Korrespondenz bewerten werden kann. Die geschieht, indem der *Euklid'scher Abstand* (Kapitel 2.3.4) zwischen dem 3D-Feature und dem ihm zugehörigen 2D-Feature für jede Korrespondenz einer Pose gemessen und aufsummiert wird (Vergleiche mit Algorithmus aus Kapitel 4.1). Man muss sich allerdings bewusst machen, dass der Begriff der Fitness an dieser Stelle nicht ganz korrekt ist. Entsteht bei dieser Berechnung ein großer Wert, bedeutet es nicht, dass das Individuum eine hohe Fitness besitzt. Man sollte diesen Wert als eine Art negative Fitness betrachten. Denn umso größer dieser Wert ist, desto schlechter ist dieses Individuum. Da im Kontext eines *genetischen Algorithmus* der Schritt der Bewertung immer als Fitnessfunktion bezeichnet wird, soll es auch hier so beibehalten werden. Eine korrekte Bezeichnung wäre beispielsweise Distanzfunktion.
Wie sich gezeigt hat, mussten geschätzte Posen mit zu großen z-Werten bestraft werden. Der Algorithmus hatte die Eigenart, Posen zu bevorzugen, die das 3D-Feature Modell weit in den Raum hinein transliert und projeziert haben. Das lässt sich damit begründen, dass mehrere 3D-Feature auf

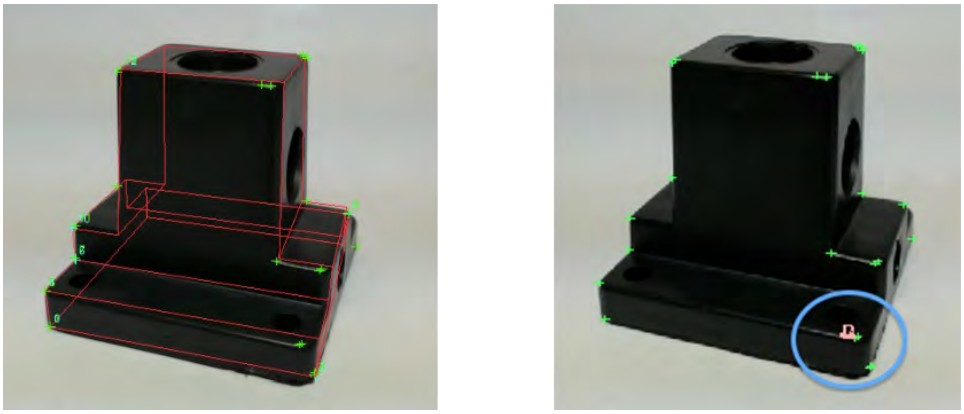

Abbildung 33: Posenvergleich von Individuen mit maximaler Fitness. Links mit Bestrafung und rechts ohne. Quelle: eigene Erstellung

ein 2D-Feature abgebildet werden können. Im Extremfall wurde der z-Wert der Translation so groß, dass einige 3D-Feature einer jeden Korrespondenz einer Pose auf ein 2D-Feature abgebildet wurden. Mit einer Fitnessfunktion ohne Anpassung würde dementsprechend solch eine Pose als sehr gut bewertet werden, da der Pixelabstand der Korrespondenzen sehr klein ist. Ein sehr gutes Individuum bedeutet auch eine hohe Reproduzierbarkeit, was nach wenigen Generationen ein Dominanzproblem auslöst. Der Algorithmus kommt aus diesem lokalen Extrema nicht mehr herauskommt. In Abbildung 33 wurde der Algorithmus mit Bestrafung (links) und ohne Bestrafung (rechts) durchgeführt. Beide dargestellten Posen besitzen eine aufsummierte Distanz von fünf Pixel, was einer sehr guten Fitness entspricht. Trotzdem sind die Ergebnisse sehr unterschiedlich. Im rechten Bild wurden nahezu alle sechs Korrespondenzen auf ein 2D-Feature abgebildet, woraus sich automatisch ein hoher z-Wert der Translation ergibt. Im linken Bild hat jede Korrespondenz ein anderes 2D-Feature.

4.4.4 Wahrscheinlichkeitsmaß

Wurde jedes Individuum mit einer Fitness versehen, ist es für die weiteren Schritte eines *genetischen Algorithmus* wichtig, alle Individuen in eine Relation zueinander zu setzen. Dazu wird als weiterer Wert die Wahrscheinlichkeit eines Individuums berechnet und ihm zugewiesen. Eine Berechnung wie in Kapitel 2.5.3.3 kann aufgrund des nicht ganz korrekten Begriffs der Fitness, wie in Kapitel 4.4.3 angegeben, nicht gemacht werden. Als eine gute Möglichkeit hat sich erwiesen, zunächst die Wahrscheinlichkeit wie in Kapitel 2.5.3.3 zu berechnen, aus ihr die umgekehrte Wahrscheinlichkeit erstellen, um einem gutem Individuum mit niedriger Fitness eine hohe Wahrscheinlichkeit zuzuteilen. Die umgekehrte Wahrscheinlichkeit lässt

sich mit Formel 4.4.1 berechnen.

$$UWsk_i = 1 - Wsk_i \tag{4.4.1}$$

Tabelle 1 verdeutlicht diesen Vorgang. Wie man sieht, ist das fünfte Individuum das Beste. Allerdings spiegelt sich das durch Berechnung nach Kapitel 2.5.3.3 nicht in der Wsk wieder (Spalte 2). Es besitzt eine sehr geringe Wahrscheinlichkeit ausgewählt zu werden. Durch die Berechnung der umgekehrten Wahrscheinlichkeit lässt sich dies korrigieren (Spalte 3).

Individuum	Fitness	Wsk	$UWsk$
1	40	0,129	0,871
2	103	0,333	0,667
3	35	0,113	0,887
4	121	0,391	0,609
5	10	0,032	0,968

Tabelle 1: Tabelle zur Verdeutlichung des missverständlichen Fitnessbegriffs.

4.4.5 Fitnessproportionale Selektion

Die fitnessproportionale Selektion wurde in dieser Arbeit anhand einer Glücksradauswahl umgesetzt. Sie ist unter anderem für ihre effiziente und einfache Implementierung beliebt. Folgender Algorithmus soll das Vorgehen verdeutlichen.

- *1. Schritt:* Erzeuge eine Zufallszahl, die im Wertebereich zwischen 0 und der aufsummierten $UWsk$ aller Individuen liegt.
- *2. Schritt:* Nimm ein noch nicht gewähltes Individuum und subtrahiere seine $UWsk_i$ von der Zufallszahl.
- *3. Schritt:* Ist die Zufallszahl kleiner oder gleich 0, dann wurde das Individuum aus dem 2. Schritt selektiert. Beende die Selektion hier. Ist die Zufallszahl größer als 0, dann mache mit dem 2. Schritt weiter.

Neben der Anpassung der Wahrscheinlichkeitsberechnung muss auch die fitnessproportionale Selektion leicht angepasst werden. Wie in Kapitel 4.4.4 beschrieben, kann man sich zunutze machen, dass die aufsummierte Wsk aller Individuen immer 1 ist. Somit könnte im ersten Schritt des vorherigen Algorithmus die Zufallszahl immer zwischen 0 und 1 gewählt werden. Allerdings stimmt diese Eigenschaft für $UWsk$ nicht mehr. Zur Verdeutlichung dient Tabelle 2. Für den *genetischen Algorithmus* bedeutet diese Ei-

Individuum	Fitness	Wsk	$UWsk$
1	40	0,129	0,871
2	103	0,333	0,667
3	35	0,113	0,887
4	121	0,391	0,609
5	10	0,032	0,968
		1.0	4,002

Tabelle 2: Tabelle zur Verdeutlichung des missverständlichen Fitnessbegriffs. Erweitert mit der aufsummierten Wahrscheinlichkeit

genschaft, dass vor einer fitnessproportionalen Selektion zwangsläufig die Summe aller $UWsk$ errechnet muss, um so den Wertebereich der Zufallszahl festzulegen.

4.4.6 Mutation

Wurde eine neue Population nach der fitnessproportionalen Selektion ausgewählt, geht es in diesem und dem nächstem Kapitel darum, neue Individuen zu erzeugen, um so den Genpool, bzw. den Suchraum zu durchsuchen.
In der Biologie gibt es mehrere Möglichkeiten, wie eine Mutation stattfinden kann. Einige wurden in Kapitel 2.5.3.7 beschrieben. Im Kontext des *genetischen Algorithmus* hat sich vor allem die Punktmutation durchgesetzt. Sie wird auch in dieser Arbeit verwendet.
Bei einer Punktmutation wird ein Gen eines Individuums verändert. Da in dieser Arbeit ein Individuum aus Korrespondenzen besteht, wird eine Korrespondenz verändert. Es gibt zwei Mglichkeiten wie eine Korrespondenz mutieren kann. Zum einen indem ein Feature, (ob 3D-oder 2D-Feauture ist an dieser Stelle egal,) durch ein anderes Feature ausgetauscht wird. Zum anderen, indem beide Feature ausgetauscht werden. Mit anderen Worten die Korrespondenz verworfen und eine neue zufällig gewählt.
Beide Verfahren wurden umgesetzt und treten mit jeweils einer eigenen Wahrscheinlichkeit ein, die über die Benutzeroberfläche separat eingestellt werden kann.

4.4.7 Crossover

Neben der Mutation bringt Crossover ebenfalls neue Individuen hervor. Wie schon bei der Mutation existieren in der Biologie diverse Mglichkeiten, wie Crossover eintreten kann. Einige solcher Möglichkeiten wurden in Ka-

pitel 2.5.3.6 angegeben. Auch bei diesem Schritt eines *genetischen Algorithmus* hat sich ein Verfahren durchgesetzt. Es handelt sich um das *One-Point-Crossover*. Ein Individuum besteht aus einem Chromosom. Dieses Chromosom ist beschrieben durch sechs Korrespondenzen. Tritt ein Crossover ein, werden die Chromosomen der Individuen A und B willkürlich an der selben Stelle durchtrennt und über Kreuz wieder zusammengefügt. Wurde beispielsweise der Schnittpunkt zwischen Gen drei und vier gewählt, dann erhält Individuum A beim Zusammenführen die Gene 4, 5 und 6 von Individuum B und Individuum B die Gene 4, 5, und 6 von Individuum A.
Dadurch entstehen zwei völlig neue Individuen, die zwar die selben Korrespondenzen wie vorher verwenden, aber neue Lösungskandidaten darstellen.

4.4.8 Insel Evolution

Aufgrund der erfolgreichen Ergebnisse, wie in Kapitel 2.5.3.9 angegeben, wurde sich auch für in dieser Arbeit dazu entschieden, den *genetischen Algorithmus* durch eine Insel Evolution zu erweitern. Der Benutzer kann in der Implementierung dieser Arbeit auswählen, wie viele Inseln benutzt werden sollen, wie lange eine Epoche dauern soll und wie viele Individuen migrieren.
Durch die Verwendung der Insel Evolution wird sich erhofft, dass der Algorithmus schneller eine gültige Pose findet und zudem auch der Computer besser ausgenutzt wird. Wie schon erwähnt, bietet es sich an, jede Insel einem eigenen Thread zuzuweisen.

4.4.9 Pest und Krankheit

Pest und Krankheit ist ein Verfahren, dass im Zuge dieser Arbeit entstanden ist und in der Literatur so nicht gefunden wurde. Es befasst sich insbesondere mit dem Dominanzproblem. Das Dominanzproblem bezeichnet das Phänomen des *genetischen Algorithmus*, das ein gutes Individuum die gesamte Population beeinflusst.
Die fitnessproportionale Selektion bietet das korrekte Verhalten, dass ein Individuum durchaus mehrfach in die nächste Generation übernommen werden kann. Existiert nun ein Individuum, das wesentlich besser ist als alle anderen, so wird dieses Individuum wahrscheinlich mehrfach in die nächste Population übernommen. Dadurch steigert sich auch seine Übernahme in die übernächste Population. Neben der Problematik der erhöhten Selektionswahrscheinlichkeit verbreitet dieses Individuum durch Crossover auch sein Erbgut an anderen Individuen. Soweit, bis nur noch dieses Erbgut im Genpool vorhanden ist. Diesem Effekt soll das Prinzip der Mutation entgegenwirken. Allerdings hat sich gezeigt, dass es nur bedingt ausreichend ist.

Aus diesem Grund wurde ein weiteres biologisches Verfahren in den *genetischen Algorithmus* dieser Arbeit aufgenommen. *Pest und Krankheit* dient dazu, den Genpool möglichst groß zu halten. Dies funktioniert, indem nach einer gewissen Anzahl Generationen ohne Verbesserung des besten Individuums, Individuen aus der Population entfernt und durch neue, zufällig gewählte Individuen ersetzt werden. Bei diesem Verfahren werden zwei Vorgehensweisen unterschieden: Zum einen das der Krankheit. Hierbei werden alle Individuen bis auf die Eliten (Kapitel 2.5.3.8) ersetzt. Ist die Anzahl der Eliten gleich null, werden alle Individuen ersetzt bis auf das Beste. Das zweite Vorgehen nennt sich Pest. Wie der Namen vermuten lässt, ist das ein drastischer Eingriff in die Population. Hierbei werden restlos alle Individuen durch Neue ersetzt. Man kann es auch mit einem Neustart eines *genetischen Algorithmus* vergleichen. Allerdings mit dem Unterschied, dass eine Kopie des besten Individuums gesichert wird. Tritt im Laufe einer Evolution mehr als eine Pest ein, wird sich lediglich das beste Individuum gemerkt. Die Besonderheit dabei ist, dass das beste Individuum zwar gemerkt wird, aber nicht mehr Teil der Population ist und sie somit nicht mehr dominieren kann.
Ein *genetischer Algorithmus* besitzt zwei Abbruchkriterien. Das Erste ist das erreichen einer gewissen Fitness. Das Zweite ist eine maximale Anzahl Generationen. Wurde die maximale Anzahl erreicht, so wird das momentan beste Individuum mit dem besten gemerkten Individuum verglichen und das Bessere wird als Lösungsvorschlag angegeben.
Nach wie viel Generationen eine Pest oder Krankheit eintreten kann, wird in der Implementierung durch die Benutzeroberfläche eingestellt. Es hat sich allerdings gezeigt, dass es sinnvoll ist, eine Krankheit nach etwa 100 Generationen und eine Pest nach etwa 500 Generationen ohne Verbesserung anzuwenden.

4.5 Zusammenfassung

In diesem Kapitel soll nun das Konzept dieser Arbeit zusammengefasst und jeweils auf die wichtigsten Punkte eingegangen werden.
Zur besseren Verständlichkeit wird das Konzept anhand eines Flussdiagramms (Abbildung 34), das stark an die spätere Implementierung angelehnt ist, dargestellt und erläutert. Dazu wird jeder Block nun kurz beschrieben.

4.5.1 Parameter

Zunächst sollen die Eingabewerte des Algorithmus angegeben werden. Dazu zählen, neben den klassischen Parametern, zur Konfiguration des *genetischen Algorithmus*, die beiden Mengen der 2D-und 3D-Feature.
Im Laufe der Entwicklung haben sich, neben den üblichen Parametern, wie

der Anzahl der maximalen Generation, Abbruchkriterium, Crossoverrate und Mutationsrate, weitere Parameter als nützlich erwiesen. Diese werden in Kapitel 6 analysiert und daher an dieser Stelle nur kurz beschrieben.

- *Accuracy* (Genauigkeit) gibt an, bei welchem Fitnesswert der Algorithmus abgebrochen werden soll.
- *Crossover* gibt an, mit welcher Wahrscheinlichkeit ein Individuum einem Crossover unterzogen wird. Diese Wahrscheinlichkeit bezieht sich auf das gesamte Individuum.
- *Disease* (Krankheit) ist ein Wert, der angibt, nach wie vielen Generationen ohne Verbesserung des besten Individuums die Methode der *Krankheit* ausgeführt wird.
- *Elites* (Eliten) gibt die Anzahl der besten Individuen an, die automatisch in die nächste Generation übernommen werden.
- *Epoch* (Epoche) beschreibt die Länge einer Epoche im Kontext der Insel Evolution. Sie gibt an, nach wie vielen Generationen ohne Verbesserungen eine Migration zwischen den Inseln stattfindet.
- *Generations* (Generationen) gibt die Anzahl der maximalen Generationen an, nach der der Algorithmus terminieren soll. Dieser Parameter beschreibt neben der *Accuracy* das zweite Abbruchkriterium.
- *Individuals* (Individuen) gibt die Menge der Individuen an, die in einer Population existieren sollen. Diese Anzahl ist konstant und verändert sich auch nicht beim Eintreten einer Pest oder Krankheit.
- *InlierMutation* ist ein Wert der angibt, mit welcher Wahrscheinlichkeit die Methode des *Beidseitigen Austauschs* eintritt. Das Eintreten dieser Methode wird für jedes Individuum getestet.
- *InlierQuantity* gibt die Anzahl der zu verwendenden Korrespondenzen an und hat somit direkten Einfluss auf die Erscheinung eines Individuums.
- *Islands* (Inseln) gibt die Anzahl der zu verwendenden Inseln an und entscheidet auch, ob eine Insel Evolution genutzt wird oder nicht.
- *Migrants* (Migranten) gibt die Anzahl der Individuen an, die im Kontext der Insel Evolution angibt, wie viele Individuen im Falle einer Migration zwischen den Inseln ausgetauscht werden.
- *Mutation* ist ein Wert der angibt, mit welcher Wahrscheinlichkeit die Methode des *Einseitigen Austauschs* eintritt. Das Eintreten dieser Methode wird für jedes Individuum getestet.

- *Plague* (Pest) ist ein Wert, der angibt, nach wie vielen Generationen ohne Verbesserung des besten Individuums die Methode der *Pest* ausgeführt wird.

2D-Feature

Als weiterer notwendiger Übergabeparmeter sind die 2D-Feature genannt. Sie können aus Bildern auf unterschiedlichste Arten gewonnen werden.

3D-Feature

Als letzter Übergabeparmeter sind die 3D-Feature notwendig. Sie können, wie in 2.2 erwähnt, ebenfalls auf unterschiedlichste Arten gewonnen werden. Die geringe Anzahl an notwendiger Information spiegelt das Ziel dieser Arbeit wieder, einen möglichst allgemeingültigen Ansatz zu finden. Es sind lediglich eine Menge an 3D-Feature und 2D-Feature notwendig.

Erstellung der Individuen

In diesem Schritt wird der *genetische Algorithmus* vielmehr die Population, mit Individuen gefüllt. Dabei werden je nach *InlierQuantity*-Parameter eine Menge an 2D-3D-Korrespondenzen zufällig erstellt.

Bewertung Individuen

Dieser Schritt ist das Herzstück des *genetischen Algorithmus*. Hier wird der komplette Bewertungsschritt eingeleitet, der jedem Individuum eine Fitness und auch Wahrscheinlichkeit zuteilt.

Pose schätzen

Aus den zuvor zufällig gewählten Korrespondenzen wird nun für jedes Individuum die entsprechende Pose geschätzt.

Projektion/Translation

Anhand der geschätzten Pose kann nun jedes 3D-Feature einer Korrespondenz auf Bildkoordinaten abgebildet werden. An dieser Stelle sein angemerkt, dass die Projektion anhand gegebener intrinsischer Kameraparameter erfolgt.

Euklid'scher Abstand

Zum Zeitpunkt dieses Schrittes, besitzt jedes Individuum eine Menge an 2D-3D-Korrespondenzen und zu jedem 3D-Feature die dazugehörige 2D-Abbildung. Für jede Korrespondenz wird der Pixelabstand zwischen 2D-

Feature und dem abgebildeten 3D-Feature gemessen und aufsummiert. Dieser Wert wird als Fitness dem Individuum zugewiesen.

Selektion

In diesem Schritt werden die Individuen anhand ihrer Fitness in die nächste Generation übernommen. Wurde ein Wert größer Null für den *Elites*-Parameter angegeben, werden zunächst die Eliten in die nächste Generation übernommen. Die restlichen Individuen werden durch eine fitnessproportionale Selektion ausgewählt.

Crossover

Crossoover ist ein Arbeitsschritt, der nur zu einer bestimmten Wahrscheinlichkeit eintritt. Die Wahrscheinlichkeit wird im Parmeter *Crossover* angegeben. Tritt ein Crossover für ein Individuum ein, wird zufällig ein weiteres Individuum ausgewählt und ihr Erbgut (Korrespondenzen) an einer zufälligen Stelle über Kreuz ausgetauscht.

Mutation

Auch die Mutation tritt nur mit einer im Parameter *Mutation* angegeben Wahrscheinlichkeit ein. Sie wird hingegen dem *Crossover* für jede Korrespondenz überprüft. Tritt sie ein, wird zwischen einem *Beidseitigen Austausch* und *Einseitigen Austausch* unterschieden. Welcher *Austausch* stattfindet, wird über den *InlierMutation*-Parameter angegeben, der die Wahrscheinlichkeit angibt, ob ein *Beidseitiger Austausch* stattfindet.

Beidseitiger Austausch

Bei dieser Art der Mutation wird die Korrespondenz komplett ausgetauscht.

Einseitiger Austausch

Wie der Name es vermuten lässt, wird nur eine Seite der Korrespondenz ausgetauscht. In der Implementierung dieser Arbeit ist es immer der 2D-Feature, der ausgetauscht wird.

Überprüfung der Abbruchkriterien

Wurde die maximale Anzahl an Generationen, angegeben im *Generations*-Parameter, erreicht oder hat das beste Individuum eine Fitness, die besser ist als der *Accuracy*-Parmeter, dann wird das momentan beste Individuum als Lösungskandidat vorgeschlagen. Viel mehr wird die geschätzte Pose dieses Individuums vorgeschlagen.

Krankheit

Hat sich nach einer gewissen Anzahl Generationen das beste Individuum nicht verbessert, tritt eine Krankheit ein. Die Anzahl der Generationen wird im *Epoch*-Parameter angegeben. Dabei werden alle Individuen, bis auf die Eliten, ausgetauscht. Wurde die Anzahl der Eliten mit Null angegeben, wird das beste Individuum automatisch übernommen.

Pest

Beim Eintreten einer Pest werden alle Individuen einschließlich der Eliten ausgetauscht. Zu welchem Zeitpunkt eine Pest eintritt, wird im *Plague*-Parameter angegeben. Bei diesem Ansatz ist es bedeutsam zu verstehen, dass das beste Individuum zwar ebenfalls aus der Population genommen wird, aber seine Kodierung wird sich gemerkt. Wurde das Abbruchkriterium der maximalen Generationen erfüllt, wird das beste Individuum, das zur gesamten Laufzeit des *genetischen Algorithmus* existiert hat, als Lösung vorgeschlagen. Es kann also auch ein Individuum sein, das eigentlich durch eine Pest entfernt wurde.

Ausgabe der besten Pose

Es wird die geschätzte Pose des besten Individuums ausgegeben.

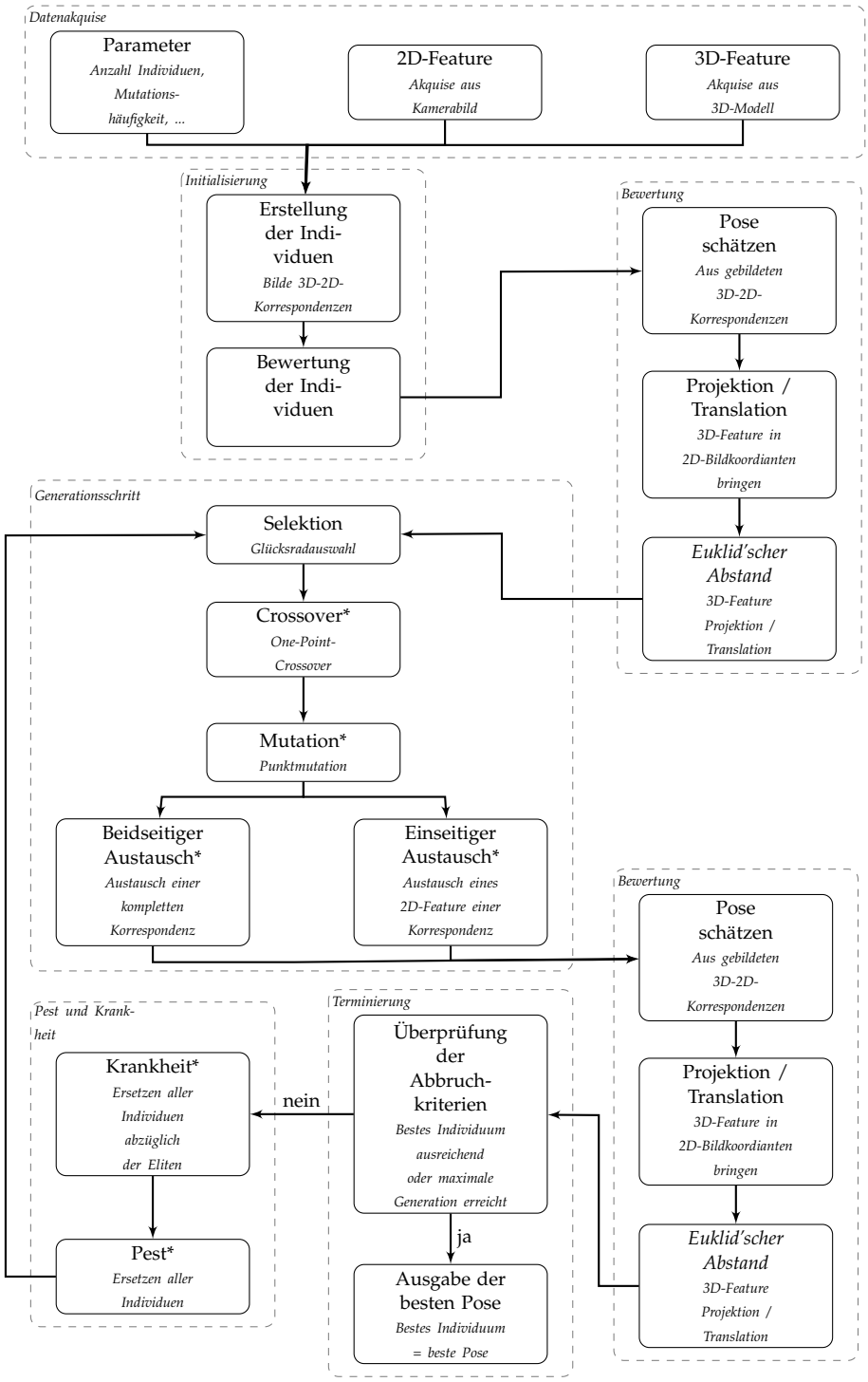

Abbildung 34: Flussdiagramm des Konzepts in Anlehnung an die Implementierung. Bereiche mit einem * werden nur zufällig oder nach Eintreten eines Ereignisses aktiv. Quelle: eigene Erstellung

5 Realisierung

In diesem Kapitel soll nun vorgestellt werden, wie das theoretische Vorgehen des Lösungsansatzes, wie in Kapitel 4.1 angegeben, implementiert wurde. Ziel war es, ein Plug-in[22] für ein bestehendes Softwareprodukt zu erstellen. Bei dem Softwareprodukt handelt es sich um die *Instant Vision*, vorgestellt in Kapitel 2.7. Das Plug-in wird von nun an als *InitialPoseAction* bezeichnet.

5.1 Verwendete Software

Die verwendete Software soll an dieser Stelle kurz beschrieben werden. Wie einleitend erwähnt, soll ein Plug-in für die *Instant Vision* entstehen. Im Kontext der *Instant Vision* werden Plug-ins auch als *Actions* bezeichnet. Die Bezeichnung als *Action* soll von nun an beibehalten werden. Die Verwendung der *Instant Vision* hat den großen Vorteil, dass sie bereits einige *Actions* mitbringt, die zum Lösen des Problems sehr nützlich sind. Dazu zählen unter anderem:

- *VideoSourceAction* stellt eine Verbindung zu einer angeschlossenen Kamera her und repräsentiert das aufgenommene Bild als einen eindimensionalen Vektor, mit dem in einer Action ohne weitere Verarbeitungsschritte gearbeitet werden kann.
- *FastCornerDetect* beinhaltet eine effiziente Implementierung des *FAST*-Algorithmus zur Punktdetektion (vorgestellt in Kapitel 2.1.3). Als Eingabe benötigt er lediglich ein Kamerabild. Ausgabe ist eine Menge von 2D-Bildkoordinaten, an denen der Algorithmus Ecken detektiert hat. Eine Verbesserung durch maschinelles Lernen (Kapitel 2.1.3.2) bietet diese Implementierung nicht. Sie wäre zum Lösen der Aufgabenstellung ohnehin nicht nützlich, da ein möglichst allgemeingültiger Ansatz gefunden werden soll. Dazu zählt auch der Verzicht auf Trainingsbilder, wie sie bei der Verbesserung durch maschinelles Lernen notwendig gewesen wären.

[22] *Wikipedia(21.04.2013):* "Ein Plug-in (deutsch etwa "Erweiterungsmodul") ist ein Softwaremodul, das von einer Softwareanwendung während seiner Laufzeit entdeckt und eingebunden werden kann, um dessen Funktionalität zu erweitern."

- *WidgetActionPipe* bietet die Möglichkeit, ein Fenster zum Anzeigen von Informationen darzustellen. In unserem Fall wird in solch einem Fenster die gefundene Pose des *genetischen Algorithmus* anhand eines 3D-Linienmodells dargestellt. Weiter bietet es auch die Möglichkeit, mehrere Informationen übereinander darzustellen. Für diese Arbeit bedeutet das, dass man das translierte 3D-Linienmodell und das Kamerabild übereinander darstellen kann und so die Pose optisch bewerten kann.

- *PointPoseAction* ist eine Action, die intern während der Fitnessberechnung aufgerufen wird. Sie bietet die Funktionalität, aus einer Menge von 2D-3D-Korrespondenzen, eine Pose zu schätzen. Dies geschieht durch eine lineare Methode (Kapitel 2.4.1).

- *CameraPerspective* erstellt eine virtuelle Kamera und bietet einige Methoden an. Die verwendete Methode dieser Arbeit ist das Translieren und Projizieren der 3D-Feature auf Bildkoordinaten. Sie benötigt zu diesem Schritt die intrinsischen und extrinsischen Kameraparameter, die für die Bildakquise genutzt wurden. Diese *Action* wird ebenfalls intern aufgerufen.

Wie zuvor beschrieben, benötigt die *CameraAction* intrinsische Kameraparameter. Diese wurden manuell mit der Software *GML Camera Calibration Toolbox* errechnet. Das Ergebnis dieser Kalibrierung ist eine XML-Datei mit den intrinsischen Kameraparametern. Die Datei wird von der *Instant Vision* unterstützt und kann in wenigen Schritten in das Programm eingebunden und an *Actions* zur weiteren Verwendung übergeben werden.

5.2 Anforderungen

Da eine *Action* für die *Instant Vision* erstellt wird, profitiert die Implementierung dieser Arbeit von den Vorzügen dieses großen Softwareprodukts. Dazu zählt die Plattformunabhängigkeit [23] und die Unterstützung von mobilen Systemen. Speziell wird die Möglichkeit angeboten, eine Anwendung für eine mobile Plattform zu erstellen und die eigentlichen Berechnungen auf einem entfernten Computer auszuführen. Damit ist es möglich, den Algorithmus augenscheinlich auf einem mobilen Endgerät laufen zu lassen, wobei die aufwendigen Berechnungen auf einem entfernten Computer, Server oder sogar Cluster ausgeführt werden. Auf diese Möglichkeit wird in dieser Arbeit nicht weiter eingegangen und sei an dieser Stelle nur kurz verwiesen.
Aufgrund des hohen Rechenaufwands des *genetischen Algorithmus* und der

[23] *Wikipedia (21.04.2013):* "Die Plattformunabhängigkeit, genauer als plattformübergreifend wird in der Informationstechnik die Eigenschaft genannt, wenn ein Programm auf verschiedenen Plattformen ausgeführt werden kann."

Verwendung einer Insel Evolution wurde entschieden, sich die Grundkenntnisse des Multithreading [24] anzueignen und umzusetzen. Der Realisierung dient die C++ Bibliothek *boost*, genauer *Boost.Thread*.
Ziel war es, das Konzept möglichst effizient und robust in der *Instant Vision* zu realisieren.

5.3 Implementierung

Da die *Instant Vision* größtenteils in C++ geschrieben ist, wurde auch die *InitialPoseAction* in dieser Programmiersprache umgesetzt. Es wurde darauf geachtet, einen möglichst wiederverwendbaren Programmcode zu erstellen, wie auch die gesamte Action objektorientiert anzulegen.
Wie in Abbildung 35 zu erkennen ist, wurden insgesamt neun Klassen für die Realisierung des Konzepts definiert. Die funktional wichtigsten sind die Klassen *VISInitialPose*, *VISEvolution*, *VISPopulation* und *VISIndividual*. Auf diese Klassen soll nun kurz eingegangen werden.

- *VISInitialPose* stellt den gesamten Rahmen der *InitialPoseAction* dar. Diese Klasse nimmt die Eingabedaten entgegen und veröffentlicht nach jeder Generation die beste Pose über das Dataset. Weiter steuert sie maßgeblich den *genetischen Algorithmus*. Sie ruft die Funktion zur Initialisierung (*VISEvolution::initEvolution()*) und zum Ausführen einer Generation (*VISEvolution::runEvolutionToTheNextGeneration()*) auf.

- *VISEvolution* ist hauptsächlich für die Verwaltung des *genetischen Algorithmus* zuständig, löst eine Pest oder Krankheit aus und besitzt die Implementierung der Selektion inklusive der Glücksradauswahl (*VISEvolution::playRoulette*).

- *VISPopulation* beinhaltet die Individuen und besitzt die Implementation der Mutation und Crossover.

- *VISIndividual* besitzt die Implementierung der Fitnessfunktion (*VISIndividual::calcFitness()*) und somit ist das Kernstück dieser Arbeit.

VISInitialPose ist die einzige Klasse, die die abstrakte Klasse *VISAction* implementiert. Im Sinne der *Instant Vision* ist sie somit die eigentliche *Action*. Die restlichen Klassen sind Hilfsklassen und implementieren den *genetischen Algorithmus*. Wie in Kapitel 2.7 genannt, implementiert eine Action die zwei Methoden *init()* und *apply()*.
Die *init()*-Methode liest die Eingabedaten ein und repräsentiert die 2D- und 3D-Feature Mengen jeweils als einen Vektor. Außerdem initialisiert diese

[24] *Wikipedia (21.04.2013):* "Der Begriff Multithreading (auch Nebenläufigkeit, Mehrsträngigkeit oder Mehrfädigkeit genannt) bezeichnet das gleichzeitige Abarbeiten mehrerer Threads (Ausführungsstränge) innerhalb eines einzelnen Prozesses oder eines Tasks (ein Anwendungsprogramm)."

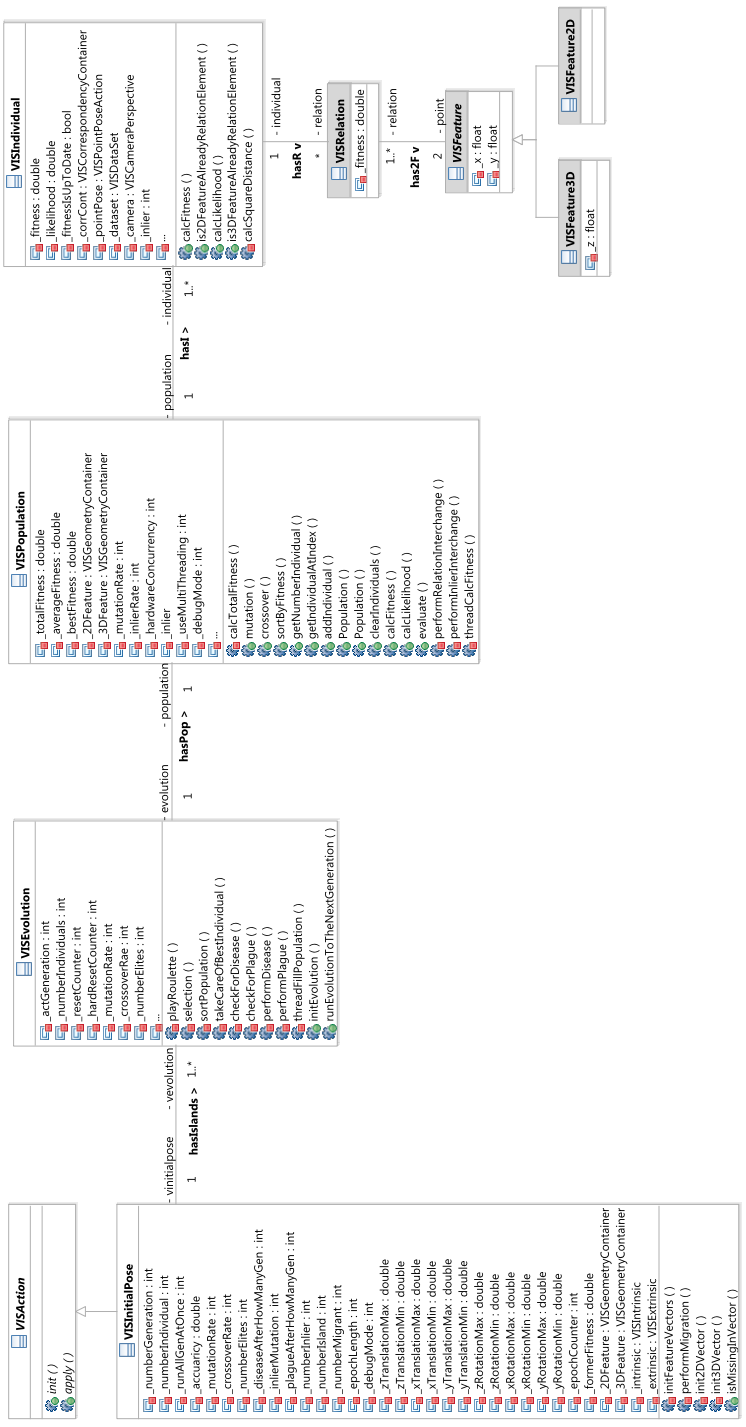

Abbildung 35: Klassendiagramm zur Implementierung. Quelle: eigene Erstellung

Methode den *genetischen Algorithmus*. Dabei werden alle Individuen erstellt und für jedes Individuum die Fitness berechnet.
Durch den Aufruf der *apply()*-Methode wird die Population, sofern das beste Individuum noch keine ausreichende Fitness besitzt oder die maximale Generation noch nicht erreicht ist, in die nächste Generation gebracht. Dabei wird zunächst eine Selektion, Crossover und Mutation durchgeführt. Die Fitness wird dann nur bei denjenigen Individuen berechnet, die einem Crossover oder Mutation unterlegen waren. Dazu wird die Variable *_fitnessIsUpToDate* überprüft.

5.3.1 Multithreading

Wie bereits erwähnt, wurde eine Insel Evolution implementiert, die auf jeder Insel einen separaten *genetischen Algorithmus* laufen lässt, wobei jede Insel in einem eigenen Thread läuft. Wurde keine Insel Evolution gewählt, beziehungsweise die Anzahl der Inseln auf Eins gesetzt, so werden bestimmte Programmteile durch Threads parallelisiert. Zu diesen Programmteilen zählen die Fitnessberechnung (*VISPopulation::threadCalcFitness()*), die zufällige Erstellung der Population (*VISEvolution::threadFillPopulation()*) und die Selektion.

6 Evaluation

In diesem Kapitel soll das vorgestellte Konzept evaluiert werden. Dazu wird zunächst die im vorherigen Kapitel beschriebene Implementierung aus Sicht des Benutzers erläutert. Als nächstes wird der Aufbau des Versuchs erklärt. Dabei geht es zum einen darum, unter welchen Umständen das Kamerabild aufgenommen wurde (äußere Umstände) und zum anderen welche Hardware genutzt wurde (innere Umstände). Im Anschluss wird angegeben, wie eine Versuchsreihe aufgebaut ist und welche Versuche gemacht wurden. Deren Ergebnisse werden dann im Kapitel 6.4 vorgestellt.

6.1 Funktionen

Die Implementierung erscheint dem Benutzer innerhalb der Oberflche der *Instant Vision*.
An dieser Stelle muss der Begriff des Benutzers im Kontext dieser Software allerdings genauer erläutert werden. Die *Instant Vision* ist nicht als Software für den Endbenutzer gedacht. Vielmehr ist sie eine Sammlung an Algorithmen (*Actions*), die von einem Entwickler zu einer Sequenz *Actionpipe* zusammengefügt werden kann. Unter Zuhilfenahme von VRML oder X3D[25] kann daraus ein lauffähiges Programm für den Endbenutzer erstellt werden, das beispielsweise mit dem *Instant Player*[26] ausgeführt wird. Ein eigenständiges Programm zu entwickeln wäre zwar möglich, wird aber nur selten gemacht.
Die *Instant Vision* präsentiert sich dem Benutzer, wie in Abbildung 36 farblich hervorgehoben, in fünf Bereichen.

- *blau:* Der blaue Bereich markiert die grafische Darstellung der *Actionpipe* mit ihren *Actions*. Sie werden sequentiell von oben nach unten abgearbeitet.

- *gelb:* Das gelb markierte Fenster erscheint nach einem Doppelklick auf eine *Action* und bietet dem Benutzer die Möglichkeit, Attribute

[25] *Wikipedia (23.04.2013):* "Extensible 3D, kurz X3D, ist eine auf XML (XML-Encoding, Datei-Endung .x3d) basierende Beschreibungssprache für 3D-Modelle, die in einem Webbrowser angezeigt werden können."
[26] Ebenfalls Teil der *VisionLib*

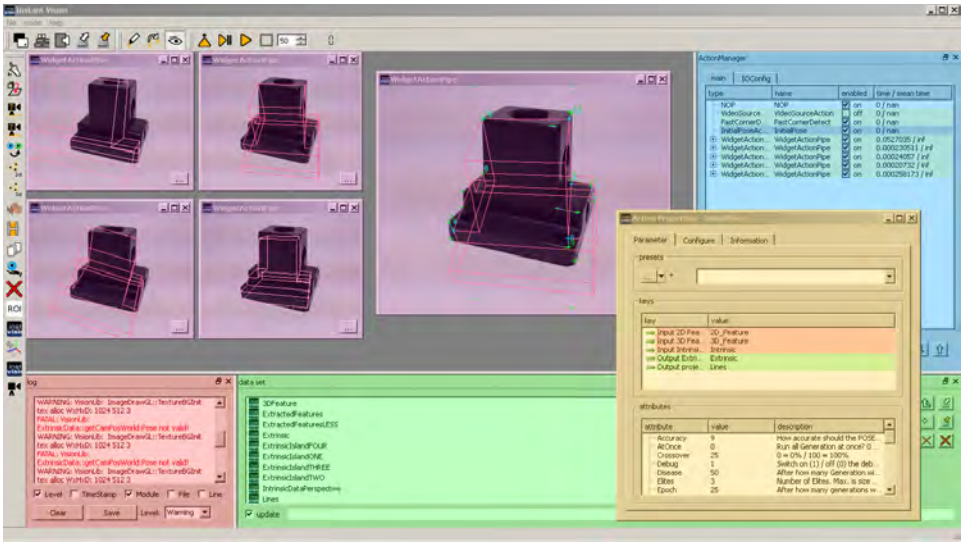

Abbildung 36: Implementierung aus Sicht des Benutzers. Die Benutzeroberfläche der *Instant Vision*. Quelle: eigene Erstellung

und Parameter anzupassen: in dieser Abbildung am Beispiel der *InitialPoseAction* gezeigt.

- *grün:* Die grüne Hinterlegung hebt die grafische Darstellung des *Datasets* hervor. Über dieses tauschen alle *Actions* Daten aus.
- *rot:* Der rote Bereich ist das Log und bietet der *Instant Vision* die Möglichkeit mit dem Benutzer zu kommunizieren, um auf fehlerhafte Daten hinzuweisen oder sonstige Nachrichten auszugeben.
- *magenta:* Die magenta markierten Fenster werden genutzt, um Daten aus dem *Dataset* anzuzeigen. Sie stehen in der Regel am Ende der *Actionpipe* und sind über den Namen *WidgetActionPipe* zu identifizieren.

Wie im Attributsfenster (gelb markiert) teilweise zu sehen ist, hat der Benutzer Einfluss auf alle Parameter des *genetischen Algorithmus*, die in Kapitel 4.5.1 angegeben wurden. Diese gilt es, sofern es sinnvoll erscheint, im weiteren Verlauf dieses Kapitels zu evaluieren.

6.2 Versuchsaufbau

Der Versuchsaufbau wurde aufgrund der hohen Komplexität des zu lösenden Problems so gewählt, dass möglichst gute Eingabedaten erhalten werden. Wie einleitend zu dieser Arbeit bereits beschrieben, gibt es keinen so allgemeinen Ansatz, der auf Benutzereingaben und reichhaltiges Vorwissen der Szenerie verzichtet. Aus diesem Grund wurde der Versuchsaufbau möglichst einfach gewählt, um das Hauptaugenmerk auf die Idee dieser

Arbeit zu lenken.
Das Objekt, von dem die Pose gefunden werden soll, ist in den Versuchen ein Lampenständer, der dazu dient, eine Bürolampe an einer Tischplatte zu befestigen. Wie in Abbildung 37 gut zu erkennen ist, besitzt er markante Ecken und bis auf die kreisrunden Durchbohrungen keine überflußigen Details. Ein weiterer Vorteil dieses Objekts ist die geometrische Einfachheit, die es ermöglichte in kurzer Zeit ein 3D-Modell zu erstellen.

Abbildung 37: Im linken Bild ist das Objekt, das zur Evaluierung genutzt wird, zu sehen und im rechten Bild das dazugehörige 3D Linienmodell. Quelle: eigene Erstellung

Der Lampenständer wurde dazu, wie in Abbilung 38 gezeigt, vor einer weiße gekrümmte Wand positioniert. Durch die Krümmung konnte man Bilder aufnehmen, die den *FAST-Punktdetektor* nur am Objekt anschlagen lassen, da es keinen störenden Hintergrund gibt oder Ecken im sonstigen Bild zu finden sind. Außerdem hat sich gezeigt, dass die verwendete Webcam bei nicht optimalen Lichtverhältnissen verrauschte Daten liefert. Das führt bei dem verwendeten Punktdetektoren zu Feature, die über das gesamte Bild gleichmäßig verteilt sind. Aus solch einem Datensatz lässt sich natürlich ohne weiteres keine Struktur des Objekts entnehmen. Zum einen könnte man das Bild mit einem Weichzeichnungsfilter bearbeiten, allerdings würden dabei auch Informationen des gesuchten Objekts verloren gehen (Vergleich Kapitel 4.1). Um das Rauschen der Kamera, aus den genannten Gründen zu reduzieren, wurde das Objekt mit einer Lichtquelle ausreichend ausgeleuchtet. Gut zu erkennen ist das in Abbildung 38 rechts. Außerdem bietet ein künstlich ausgeleuchteter Versuchsaufbau den Vorteil, dass zu jeder Tageszeit gleiche Bedingungen vorherrschen. Dies vereinfacht die Evaluation.
Als Webcam wurde eine *Logitech C920* genutzt und zum Ausführen des Algorithmus wurden insgesamt drei Computer separat genutzt. Zwei Computer sind vom Typ *Precision* der Firma *Dell*. Sie besitzen jeweils 8 GB-RAM, *Intel* Xeon i7 CPU mit 4×2,4Ghz, als GPU eine Nvidia Quadro und einer klassischen HDD. Der dritte Rechner ist ein Apple MacBook Pro. Auf

Abbildung 38: Versuchsaufbau zur Evaluierung des Lösungsansatzes dieser Arbeit. Quelle: eigene Erstellung

diesem Rechner wurde *Windows 7* virtualisiert, da der institutsinterne Support für Windows gegeben wurde. Er besitzt folgende Daten: 8 GB-RAM, *Intel* Core-2-Duo CPU mit 2×2,8Ghz, als GPU eine Nvidia GeForce 9600GT und eine klassische HDD. Da sich herausstellte, dass das Finden einer gültigen Pose je nach Parameter lange dauern kann, wurde sich dazu entschieden, mit drei Computern zu evaluieren, um im Rahmen der zeitlichen Vorgabe der Arbeit genügend Versuchsdaten zu ermitteln. Jeder Computer führt einen Versuchsdurchlauf aus und ist unabhängig gegenüber den anderen beiden.
Abschließend sei noch erwähnt, dass nach jeder Generation des *genetischen Algorithmus* ein grobes Abbild der Population in Form einer Textdatei auf einen zentralen Computer über das Netzwerk gesendet wurde. Dieses Abbild umfasst folgende Daten:

- Einmalig zu Beginn der Laufzeit die Angabe aller Parameter
- Angabe der aktuellen Generation
- Angabe der Fitness des besten Individuums
- Angabe der durchschnittlichen Fitness der gesamte Population
- Angabe der Insel, auf der sich das beste Individuum befindet

6.3 Versuchsdurchführung

In diesem Kapitel soll kurz beschrieben werden, welche Versuche durchgeführt wurden und auf welchen Datensätzen die Ergebnisse beruhen werden.
Jeder Versuch wurde insgesamt zwölf Mal ausgeführt. Es wurde darauf geachtet, obwohl es keine größere Auswirkungen hat, dass ein Versuch jeweils vier Mal auf jedem Computer ausgeführt wurde. Obwohl sich während der Entwicklung gezeigt hat, dass 2000 Generationen ausreichend

sind, wurde um auch schlechte Parameterwerte in Relation zu besseren setzen zu können, die maximale Generation einheitlich auf 60.000 gesetzt.
Evaluiert wird, wie sich die Parameter qualitativ auf die vorgeschlagene Lösung ausüben. Außerdem wird in einer kleiner angelegten Versuchsreihe die Abhängigkeit gegenüber texturierten Hintergründen und Hinzunahme von anderen Objekten ausgewertet.
Weiter sei erwähnt, dass ein Versuchsdurchlauf als gültig erklärt wird, sobald der Fitnesswert unter den angegeben Wert des *Accuracy*-Parameter fällt. Dieser Wert wird so gewählt, dass die Pose fast exakt sein muss. In Abbildung 39 ist auf der rechten Seite ein ungültiger und auf der linken Seite ein gültiger Durchlauf dargestellt. Als die allgemeine Güte eines Versuchsdurchlaufs wird die Trefferquote ermittelt. Also wie viele gültige Versuche es gab.

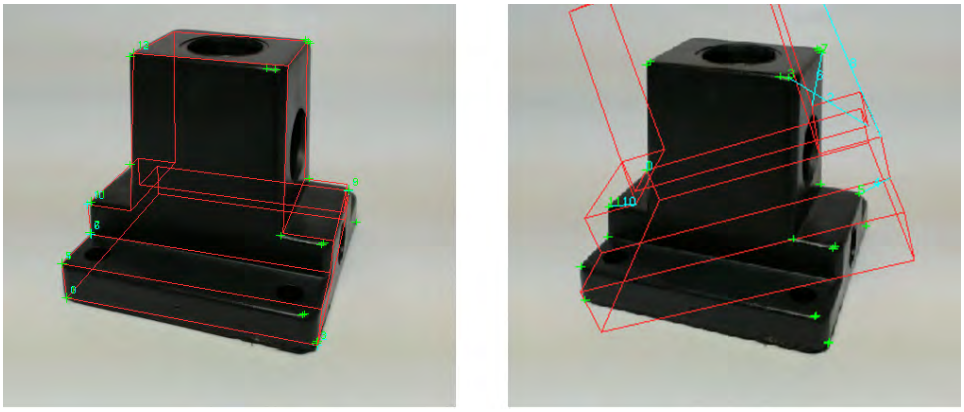

Abbildung 39: Darstellung eines gültigen (links) und ungültigen(rechts) Versuchs. Quelle: eigene Erstellung

6.4 Ergebnisse

Die Versuche lassen sich in zwei Gruppen unterteilen. Zum einen in Versuche, die die Auswirkungen der Parameter auf den *genetischen Algorithmus* bewerten und zum anderen in solche, die zeigen, welche Auswirkungen äußere Einflüsse haben.
Schwierig war es zu entscheiden, welche Startkonfiguration gewählt werden sollte. Der Algorithmus kann über eine Vielzahl von Parametern beeinflusst werden. So war es wichtig Paramater begründet auf einen Wert zu setzen, um dann Parameter nach Parameter zu evaluieren. Einerseits wurden Parameter zu Beginn auf Werte gesetzt, die in der Literatur empfohlen wurden. Andererseits, sofern in der Literatur keine Angabe gemacht worden ist, auf Werte, die sich im Laufe der Entwicklung als gut erwiesen haben. Also empirisch festgelegt. Tabelle 3 gibt an wie die Parameterwerte festgelegt wurden. Wie in der Auflistung zu sehen ist, wurden bestimm-

Parametertyp	Parameterwert	Quelle
Accuracy	9	empirisch
Crossover	25	[58]
Disease	50	empirisch
Elites	1	[58]
Epoch	-	-
Generations	60.000	empirisch
InlierMutation	2	empirisch
InlierQuantity	7	empirisch
Islands	1	empirisch
Migrants	-	-
Mutation	1	[59]
Plague	-	-
Translation X	∞	empirisch
Translation Y	∞	empirisch
Translation Z	8 bis 15,2	empirisch
Rotation X	-190 bis -110	empirisch
Rotation Y	-40 bis 40	empirisch
Rotation Z	-40 bis 40	empirisch

Tabelle 3: Tabelle der Parameterwerte zu Beginn der Evaluierung. Parameter mit einem "-"gekennzeichnet, spielen momentan keine Rolle.

te Werte mit einem "-"markiert. Diese Werte spielen momentan noch keine Rolle und werden zu einem späteren Zeitpunkt gesetzt.
Zu den drei Parametern *Islands*, *Epoch* und *Migrants* sei noch erwähnt, dass sie die Stellgrößen der Insel Evolution sind. Da wir zunächst für die Evaluierung ohne Verteilung auf verschiedene Inseln auskommen, sind diese Parameter nicht gesetzt und die Anzahl der Inseln auf Eins gesetzt. Der *Individuals*-Parameter taucht in der Tabelle im übrigen nicht auf, weil er der erste Parameter sein wird, der evaluiert wird.

6.4.1 Auswirkungen der Parameter

Als erstes werden die Ergebnisse der Evaluierung präsentiert, die die Abhängigkeit des *genetischen Algorithmus* gegenüber der Anzahl der Individu-

en darstellt.

6.4.1.1 Anzahl der Individuen

Abbildung 40 stellt die Verbindung zwischen Trefferquote und Anzahl der Individuen dar, sowie der Laufzeit einer einzelnen Generation zu der Anzahl der Individuen. Wie sich einfach ablesen lässt, steht die Anzahl der Individuen in direkter Beziehung zu der Laufzeit. Verdoppelt sich die Anzahl der Individuen, verdoppelt sich auch die Laufzeit. Hierbei sei herausgestellt, dass die Laufzeit in diesem Fall die Zeit beschreibt, die benötigt wird, um einen Generationsschritt durchzuführen. Wie solch ein Schritt aufgebaut wird, kann in Abbildung 34 nachvollzogen werden. Da dieser Wert maßgeblich von der Leistung des Computer ausgeht und durch diesen skaliert wird, soll an dieser Stelle nicht weiter darauf eingegangen werden.

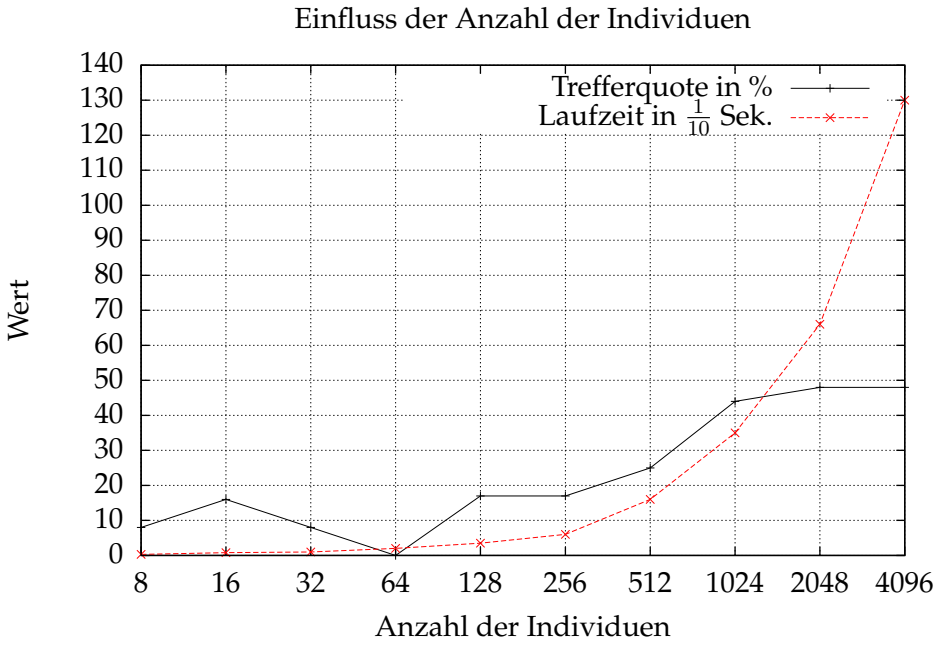

Abbildung 40: Trefferquote und Laufzeit im Vergleich zur Anzahl der Individuen. Quelle: eigene Erstellung

Interessanter zu betrachten ist die Abhängigkeit gegenüber der Trefferquote. Wie man auf den ersten Blick feststellt, hängt die Trefferquote im Großen und Ganzen von der Anzahl der Individuen ab. Der Einbruch bei der Versuchsreihe mit 32 und 64 Individuen lässt sich so erklären, dass die Individuen nicht ausreichend bei der Initialisierung verteilt wurden. Auch ist

es für ein einzelnes Individuum einfacher eine Population zu dominieren, wenn es weniger Konkurrenten gibt. Das Dominanzproblem ist bei der Versuchsreihe für 32 Individuen und extrem für 64 Individuen aufgetreten. Bei Letzterem sogar so stark, dass nicht ein einziger der zehn Versuche erfolgreich war.
Weiter lässt sich festhalten, dass ab einer bestimmten Populationsgröße keine erhebliche Verbesserung mehr stattfindet. Im Falle dieser Arbeit ab ca. 1.000 Individuen. Trotz der Vervierfachung der Populationsgröße auf 4.000, stieg die Trefferquote um lediglich vier Prozentpunkte. Dieses Phänomen wurde auch in [33] beobachtet. In dieser Quelle wird vorgeschlagen, dass die Populationsgröße auf 100 gesetzt werden sollte, da bei größeren Populationen keine merkliche Verbesserung eintrete. Im Vergleich zum Optimum (4.096 Individuen) wurde in dieser Arbeit allerdings eine dreifache Verbesserung gegenüber einer Population mit 128 Individuen erzielt. Es lässt sich festhalten, dass eine gute Populationsgröße mit ca. 1.000 anzusetzen ist. Dieser Wert wird für weitere Versuche übernommen. Besitzt man genügend Zeit oder Rechenkapazitäten, kann die Größe beliebig nach oben korrigiert werden.

6.4.1.2 Anzahl der Korrespondenzen

In Grafik 41 wird zum einen die Trefferquote und zum anderen die durchschnittliche Generation, in der eine Lösung gefunden wurde, anhand der Anzahl der Korrespondenzen dargestellt. Die Anzahl der Korrespondenzen kann über die Benutzeroberfläche mit dem *inlierQuantity*-Parameter eingestellt werden.
Interessant zu betrachten ist, dass die Anzahl der Korrespondenzen nur einen geringen Einfluss auf das Ergebnis hat. Erst ab zehn Korrespondenzen wirkt sich dieser Parameter merklich auf das Ergebnis aus. Die Wahl der Versuche hat sich aus den Eigenschaften des Testbildes ergeben. Es ist maximal möglich, elf Korrespondenz zwischen Linienmodell und Testbild herzustellen. Die Untergrenze von drei ergibt sich daraus, das die verwendete *PointPoseAction* eine Pose aus lediglich drei Korrespondenzen herstellen kann, was durch einen mathematischen Trick möglich ist, mit dem Werte doppelt in das lineare Gleichungssystem eingehen. Grundsätzlich werden aber, wie in Kapitel 2.4 beschrieben, mindestens sechs Korrespondenzen benötigt, um ohne diesen Trick auszukommen.
Ein Nachteil einer zu niedrigen Wahl der Korrespondenzen ist in Abbildung 42 zu sehen. Der Algorithmus hat den Abstand der Korrespondenzen erfolgreich minimiert, so dass das Abbruchkriterium der *Accuracy* eingetreten ist. Wie man gut erkennen kann, ist der Pixelabstand zwischen allen Korrespondenzen nahezu Null. Durch die hier implementierte Fitnessfunktion kann man diesem Phänomen nur durch mehr Korrespondenzen entgegenwirken. Eine andere Möglichkeit wäre es, in einem Nachbearbei-

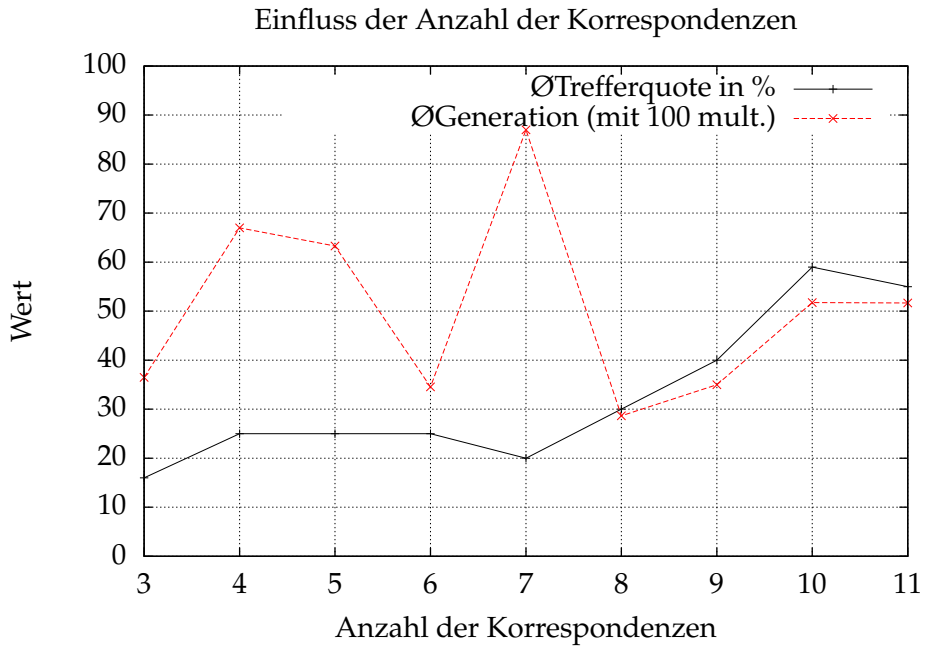

Abbildung 41: Trefferquote und Generationenanzahl im Vergleich zur Anzahl der Korrespondenzen. Quelle: eigene Erstellung

tungsschritt, die Kanten des Linienmodells mit denen des unterliegenden Objekts zu vergleichen.
Ein Versuch mit solch einem Ergebnis wurde nicht als erfolgreich eingetragen. Dies hätte die Auswertung erheblich verzerrt. Im Bereich bis zu fünf Korrespondenzen hätte die Trefferquote sonst bei nahezu 100% gelegen. Festhalten lässt sich, umso weniger Korrespondenzen, desto mehr Lösungsmöglichkeiten gibt es, was auch die Wahrscheinlichkeit erhöht, dass der Algorithmus eine Lösung findet. Der Umkehrschluss ist ebenfalls korrekt, was Abbildung 41 zeigt. Mit zehn Korrespondenzen konnten die Lösungskandidaten so eingeschränkt werden, dass der Algorithmus zu 60% erfolgreich war. Der rote Plot gibt die durchschnittliche Generation an, in der eine Lösung gefunden wurde. In dem genannten Bereich, in dem zu viele Möglichkeiten vorhanden sind, konnte nur mit Glück eine gültige Lösung gefunden werden. Nämlich in dem Fall, wenn die Individuen möglichst weit entfernt von einer falschen Lösung initialisiert wurden. Darüberhinaus verhält sich der Kurvenverlauf erwartungsgemäß: Je mehr Korrespondenzen benutzt werden, desto weniger gültige Lösungen gibt es. Umso weniger Lösungen es gibt, desto länger muss der Suchraum durchlaufen, beziehungsweise mehr Generationsschritte ausgeführt werden.
In diesem Beispiel hat sich gezeigt, dass zehn Korrespondenzen sich sehr

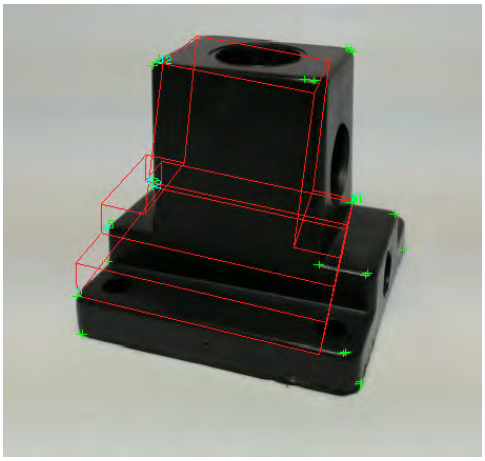

Abbildung 42: Darstellung eines gültigen Versuchs, der augenscheinlich nicht als gültig bewertet werden darf. Quelle: eigene Erstellung

positiv auf den Algorithmus auswirken. Auch dieser Wert wird für die weitere Evaluation übernommen. Abschließend lässt sich festhalten, dass die Zahl der Korrespondenzen maximiert werden sollte.

6.4.1.3 Mutationsrate

Abbildung 43 verdeutlicht, dass die vorgeschlagene Mutationsrate von einem Prozent als gut zu bewerten ist und auch in dieser Implementierung gute Ergebnisse liefert.
Es hat sich schon früh bei der Evaluierung der Mutationsrate gezeigt, dass sie nicht zu groß gewählt werden darf. Eine Erhöhung auf drei Prozent verursacht bereits ein vergleichbares Ergebnis, wie der blaue Plot in der Abbildung 43. Durch eine zu hohe Mutationsrate kann der Algorithmus nicht optimieren. Dadurch, dass die Individuen zu oft modifiziert werden, hat es die gleiche Auswirkung, als würde man in der Population Individuen durch Neue ersetzen. Dabei gehen auch optimierte Individuen verloren. Die durchschnittliche Fitness der Population kann sich somit kaum verbessern und pendelt sich schnell bei einem Wert ein. Anders ist dies hingegen bei der richtigen Wahl der Mutationsrate. Dies ist in der Abbildung am schwarzen und roten Plot zu sehen. Die durchschnittliche Fitness der Population (rot) wird kontinuierlich verbessert und folgt dem Verlauf der besten Fitness (schwarz).

6.4.1.4 Epochenlänge der Krankheit

Als Nächstes soll der Einfluss der neu entstandenen Erweiterung des *genetischen Algorithmus* evaluiert werden. In Abbildung 44 wurde zum einen

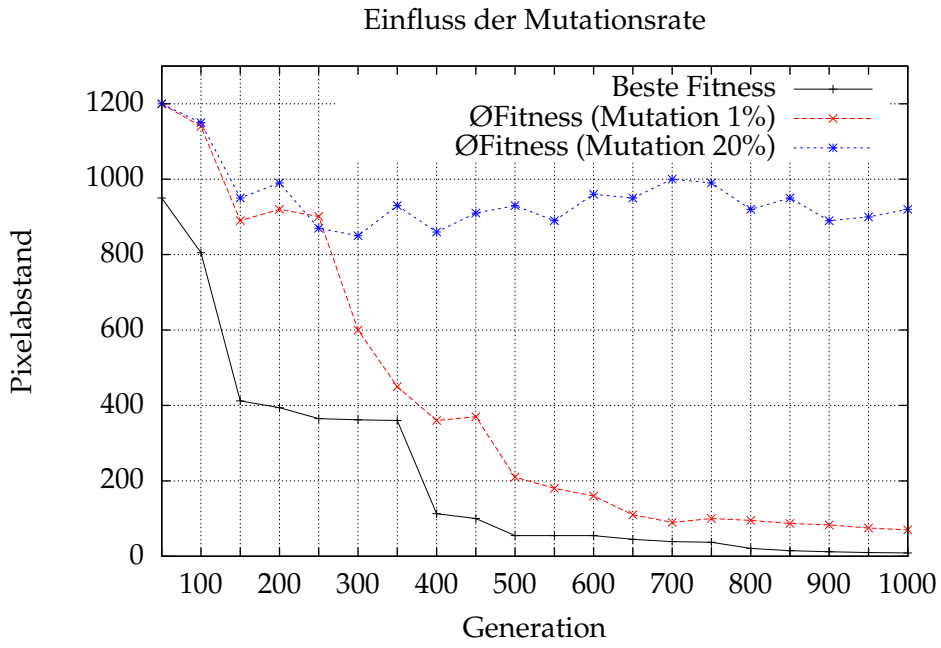

Abbildung 43: Klassischer durchschnittlicher Fitnessverlauf eines *genetischen Algorithmus* im Vergleich zu einem Entarteten. Der Verlauf des besten Individuums stammt dabei aus dem Durchlauf mit einer Mutationsrate von einem Prozent. Quelle: eigene Erstellung

die Trefferquote der momentan besten Parameter (soweit sie evaluiert wurden) ohne Krankheit angegeben. Dies ist gekennzeichnet mit "Keine" Generation. Zum anderen wurden die Ergebnisse der vier Versuchsdurchläufe, die die Epochenlänge evaluierten, geplottet.
Wie bereits in den vorherigen Versuchen gesehen, besitzt der Algorithmus mit der momentan Parameterkonfiguration eine Trefferquote von circa 50%. Da viele Stellen des Algorithmus zufallsbasiert sind, schwankt dieser Wert natürlich. In dieser Versuchsreihe konnte mit den selben Daten wie zuvor lediglich eine durchschnittliche Trefferquote von knapp unter 50% erzielt werden.
Gut zu erkennen ist, dass eine zu kurze Epochenlänge sich negativ auf die Trefferquote auswirkt. Da nach einer Epoche alle Individuen bis auf die Eliten ausgetauscht werden, verhält es sich hier wie bei einer zu hohen Mutationsrate. Der Algorithmus hat keine Chance die Population zu optimieren, weil die optimierten Individuen zu schnell durch neue Individuen ausgetauscht werden und mit sehr hoher Wahrscheinlichkeit durch Schlechtere ersetzt werden. Wie es zu vermuten war, kann dieser Parameter den Algorithmus auch negativ beeinflussen. Allerdings sieht man bei einer Länge

von 200 Generationen und darüber hinaus auch positiv. Wie beschrieben, war es Ziel dieses Ansatzes, das Dominanzproblem zu lösen. Es hat sich gezeigt, dass es nicht komplett gelöst, aber im Kontext des *genetischen Algorithmus* aufgeweicht wurde.

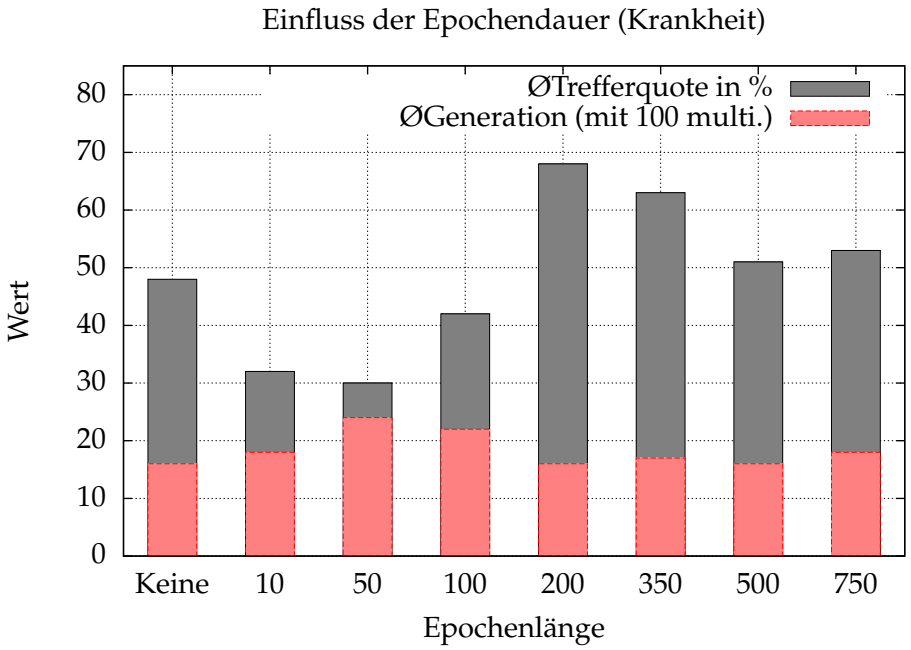

Abbildung 44: Einfluss der Krankheit und der Einfluss der Epochenlänge bis zum Eintritt einer Krankheit. Quelle: eigene Erstellung

In dieser Versuchsreihe gibt es einen Scheitelpunkt bei einer Epochenlänge von 200. Davor wird das Optimieren zu sehr behindert und darüber hinaus verhält sich der Algorithmus, als würde er ohne Krankheitsansatz arbeiten. Dies kann so begründet werden, dass das beste Individuum, auch wenn es sich in einem lokalen Extrema befindet, dort schon so stark optimiert wurde (je länger die Epoche, desto stärker), sodass es nach dem Eintreten der Krankheit wieder schnell die Population dominiert. Sofern nicht ähnlich gute Individuen erstellt werden, wird es sein Erbgut schnell verbreiten können und alle neuen Individuen infizieren.

6.4.1.5 Anzahl der Eliten

Die grafische Auswertung der Evaluation der Eliten in Abbildung 45 verdeutlicht, dass sich die Anzahl der Eliten negativ auf den Algorithmus auswirkt. Je mehr Eliten man verwendet, desto geringer ist die Wahrscheinlichkeit, dass eine Lösung gefunden wird. Außerdem dauert es länger, bis

eine Lösung gefunden wird. Dies lässt sich an der durchschnittlichen Generation ablesen. Je mehr Eliten verwendet werden, desto mehr Generationen werden benötigt, um eine Lösung zu finden.

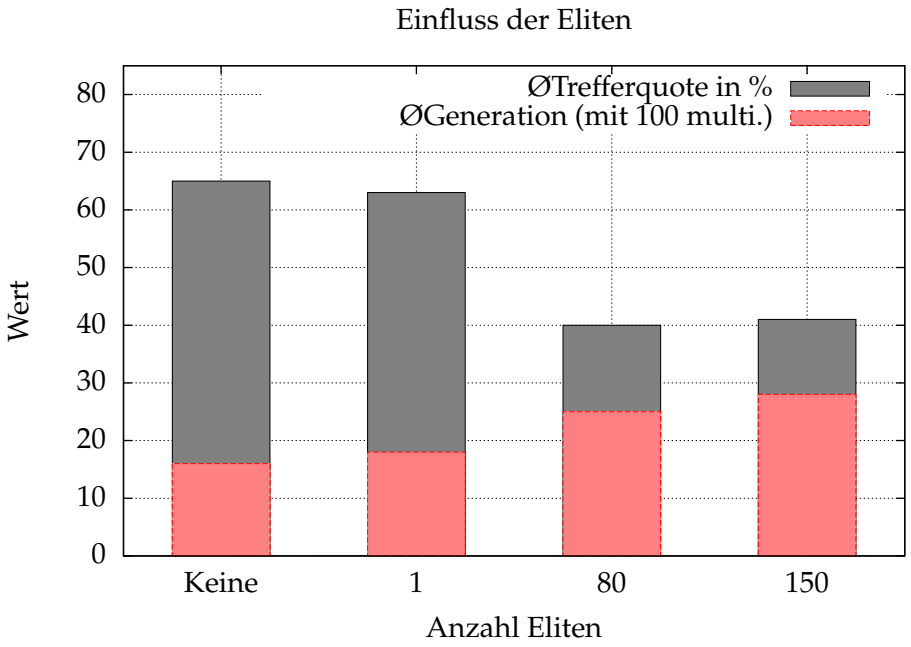

Abbildung 45: Einfluss der Anzahl der Eliten. Quelle: eigene Erstellung

Dies lässt sich so begründen, dass die besten Individuen automatisch in die nächste Generation übernommen werden. Sie werden also der *fitnessproportionalen Selektion* entzogen. Umso mehr Individuen der Selektion entzogen werden, desto geringer ist die Streuung in der nächsten Generation. Da nur noch

$$individuals_{selektion} = individuals - Elites \tag{6.4.1}$$

Individuen durch die Selektion gewählt werden. Wobei *Individuals* den Parameter der Anzahl der Individuen und *Elites* den Parameter der Anzahl der Eliten angeben.
Nehmen wir den Fall an, dass ein Individuum eine Population bereits leicht dominiert. Diesem Individuum wird durch dieses Verfahren garantiert, dass es in die nächste Generation übernommen wird. Zudem nimmt es noch an der Selektion teil, wodurch seine Dominanz weiter gefördert wird. Durch die *fitnessproportionalen Selektion* besteht die Möglichkeit, dass das beste Individuum nicht in die nächste Generation übernommen wird, was durchaus das Dominanzproblem schwächt. Durch die Verwendung von Eliten wird das Dominanzproblem allerdings gestärkt.

6.4.1.6 Einfluss der Insel Evolution

Die Insel Evolution war als eine Ergänzung gedacht, die nicht weiter eva-

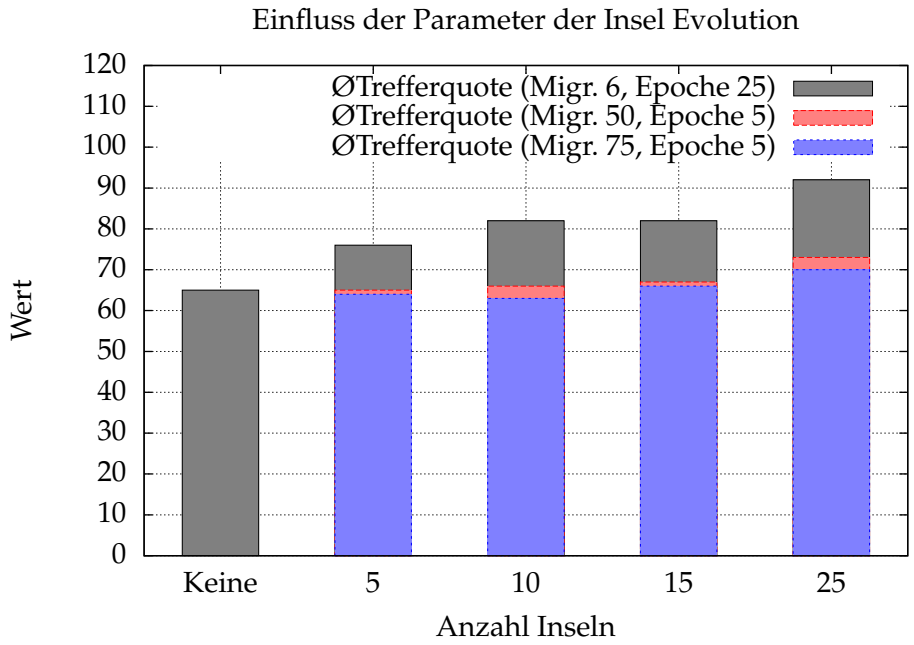

Abbildung 46: Diagramm zur Insel Evolution. Quelle: eigene Erstellung

luiert werden sollte. Das Verfahren klingt plausibel und es war nicht klar, warum es den *genetischen Algorithmus* negativ beeinflussen sollte. Trotzdem wurden zwölf Versuche durchgeführt, um den Einfluss der Parameter, die nur die Insel Evolution steuern, zu verstehen. In Abbildung 46 sind die Ergebnisse der Versuche aufgetragen. Es wurden die Parameter *Migrants* und *Epoche* bezüglich der Trefferquote untersucht. Wie man sieht, war die Annahme richtig, dass die Insel Evolution den Algorithmus positiv beeinflusst. Egal wie ihre Parameter gewählt werden. Außerdem zeigt sich, dass die Angabe in [33] zur Wahl der optimalen Parameter korrekt waren. Man sieht auch, dass eine ungünstige Wahl der Parameter, also wenn ein zu hoher Austausch zwischen den Inseln stattfindet, der Algorithmus sich verhält, als würde man die Populationgröße ohne Insel Evolution erhöhen. Da eine Vergrößerung der Population, wie bei der Evaluierung der Individuenanzahl festgestellt, auf mehr als 1.000 Individuen nur noch eine geringfügige Verbesserung aufweist, steigt auch die Trefferquote bei einer schlecht konfigurierten Insel Evolution nur wenig.
In Abbildung 47 ist dargestellt, wie eine Insel Evolution dem Benutzer angezeigt werden kann. Es wird nach jeder Generation jeweils das beste Indi-

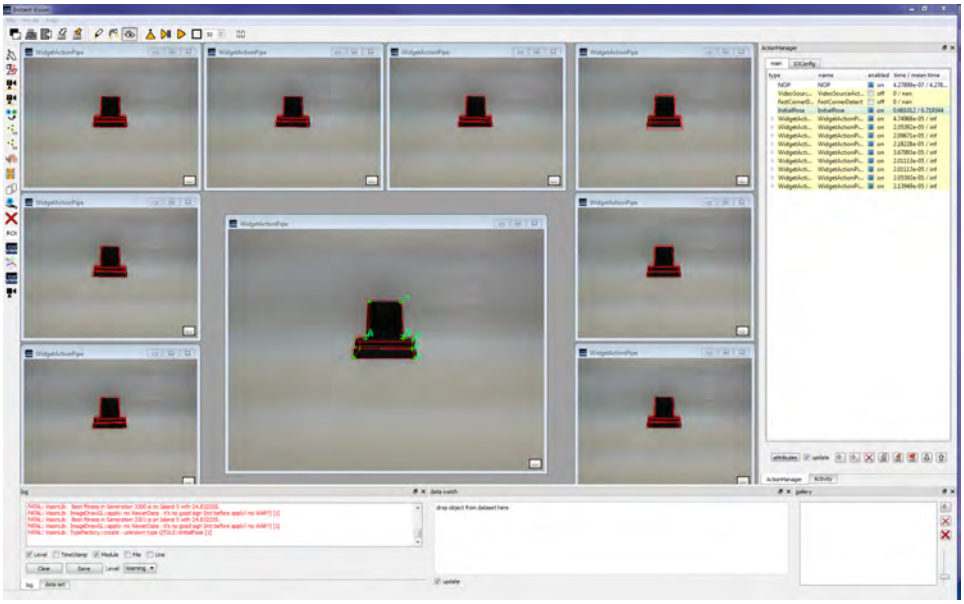

Abbildung 47: Darstellung der Benutzeroberfläche mit insgesamt acht Inseln. Im Zentrum wird das beste Individuum dargestellt. Quelle: eigene Erstellung

viduum einer Insel ermittelt und seine Pose in das Dataset gelegt. Außerdem wurde in diesem Versuch das Objekt verschoben. Ohne Anpassung der Parameter wurde das Objekt mit einer Trefferquote von über 90% gefunden.

6.4.1.7 Sonstige Parameter

Die wichtigsten Parameter wurden nun eingehend evaluiert. In diesem Kapitel sollen Beobachtungen dargelegt werden, die sich vornehmlich während der Entwicklungsphase gezeigt haben.
Der Wertebereich der Translation für X und Y spielt keine Rolle für den Algorithmus, sehr wohl aber der Z-Wert. Wie bereits in Kapitel 4.4.3 erläutert, muss hier eine Einschränkung erfolgen, damit der Algorithmus für den Betrachter richtige Lösungen ausgibt. Die Rotation sollte hingegen um alle Achsen eingeschränkt werden, um den Suchraum zu verkleinern. Allerdings reicht es bereits aus, die Rotation für X-,Y- und Z-Achse auf einen Bereich von $80°$ einzuschränken. Hingegen der Restriktion der Translation in Z-Richtung, ist diese nicht zwingend erforderlich. Ohne sie wird der Suchraum allerdings so stark vergrößert, dass die Laufzeit um ein Vielfaches ansteigt, nicht aber die Trefferquote beeinflusst.
Zum Abschluss seien noch die Parameter *Crossover*, *Accuracy* und *Plague* kurz erwähnt. *Accuracy* wird am besten so gewählt, dass für jede Korre-

spondenz ein Pixelabstand erlaubt ist. In diesem Beispiel wäre es also zehn. Dies ist als eine harte Grenze zu sehen. Hat der Algorithmus bis zu diesem Pixelabstand optimiert, liegt auch garantiert eine Lösung vor. Es hat sich gezeigt, dass eine korrekte Pose bereits mit einem doppelt so großen Pixelabstand gefunden wurde. Zur Wahl von *Accuracy* lässt sich Formel 6.4.2 festhalten.

$$Accuracy = 2 \times InlierQuantity \tag{6.4.2}$$

Crossover wurde mit nur drei Versuchen evaluiert und soll nur kurz erläutert werden. In [25] wird ein optimaler Wert mit 25% angegeben. Es wurde mit den drei Versuchen dieser Wert auf 5%, 10% und 50% gesetzt. Je weiter der Wert von 25% abweicht, desto schlechter wurde die Trefferquote. *Plague* wurde nicht evaluiert, da sich bereits während der Entwicklung gezeigt hat, dass der erhoffte Erfolg ausblieb. Im Prinzip ist es wie ein Neustart, bei dem der Algorithmus wieder von vorne anfängt.

6.4.1.8 Aktualisierung der Parameter

Aus den nun evaluierten Parametern aktualisiert sich Tabelle 3 zu Tabelle 4.

6.4.2 Auswirkungen äußerer Einflüsse

Aufgrund der begrenzten Zeit ließen sich Auswirkungen durch anderen Objekte nur noch durch einfachere Versuche untersuchen. Es wurden zwei Versuche gemacht. Ersterem wurde ein weiteres Objekt in das Bild genommen, das nur wenige neue Feature erzeugte. Der zweite Versuch hingegen war eine Szenerie mit detaillierterem Objekt, das viele neue Feature erzeugt. Wie man sehen kann, wurde der erste Versuch erfolgreich durchlaufen (Abbildung 48), Versuch zwei hingegen nicht erfolgreich (Abbildung 49). Dies lässt sich damit erklären, dass das blaue Taxi sehr viele neue 2D-Feature erzeugt hat. Umso mehr 2D-Feature, desto mehr Möglichkeiten gibt es, Korrespondenzen zu bilden (Vergleich Formel 4.2.5). Würde die Zeit keine Rolle spielen, so würde auch der letzte Versuch erfolgreich verlaufen. Es ist lediglich der Suchraum größer geworden.

6.4.3 Laufzeit

Die Laufzeiten des Algorithmus hängen maßgeblich von der Größe der Population ab, aber auch, wie zu vor in Kapitel 6.4.2 angegeben, von der Anzahl der gefundenen 2D-Feature.
Durch den Multithreading Ansatz, der einige Schritte des *genetischen Algorithmus* parallelisiert, konnte mit den neuen Parameterwerten eine Laufzeit

Parametertyp	Parameterwert	Quelle
Accuracy	9	empirisch
Crossover	25	[58]
Disease	200	evaluation
Elites	0	evaluation
Epoch	25	[60]
Generations	60.000	empirisch
Individuals	1024	evaluation
InlierMutation	2	empirisch
InlierQuantity	10	evaluation
Islands	25	evaluation
Migrants	6	[33]
Mutation	1	[59]
Translation X	∞	evaluation
Translation Y	∞	evaluation
Translation Z	8 bis 15,2	evaluation
Rotation X	-190 bis -110	evaluation
Rotation Y	-40 bis 40	evaluation
Rotation Z	-40 bis 40	evaluation

Tabelle 4: Tabelle der Parameterwerte zum Ende der Evaluierung.

von ungefähr zwölf Minuten erreicht werden. Die Posenfindung ohne Optimierung hätte nach Formel 4.2.5 und der Rechenleistung des Computers mehrere Jahre gedauert. Damit wurde ein Ansatz vorgeschlagen, der den Suchraum der Posen effizient durchsucht.

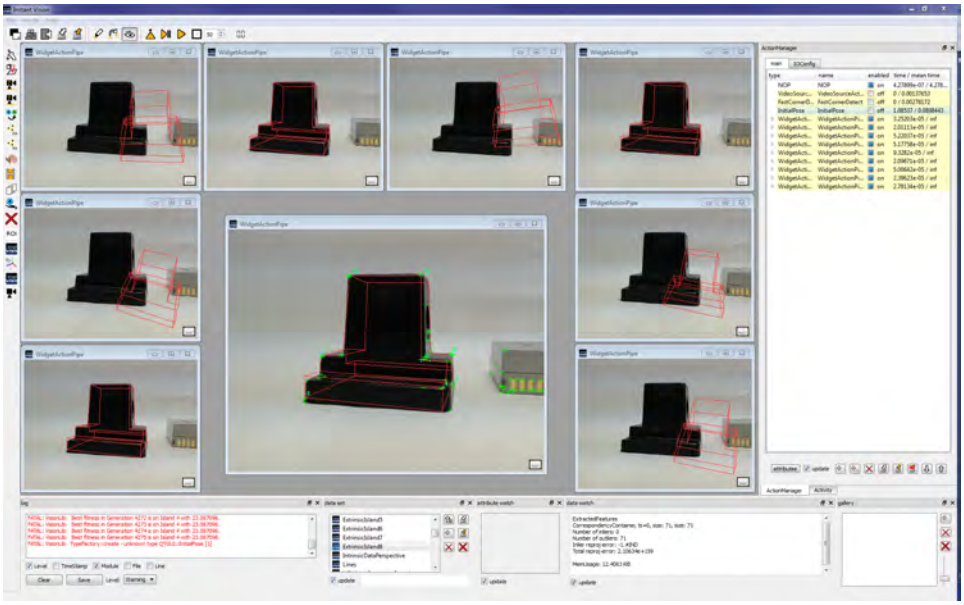

Abbildung 48: Hinzunahme eines einfachen Objekts. Quelle: eigene Erstellung

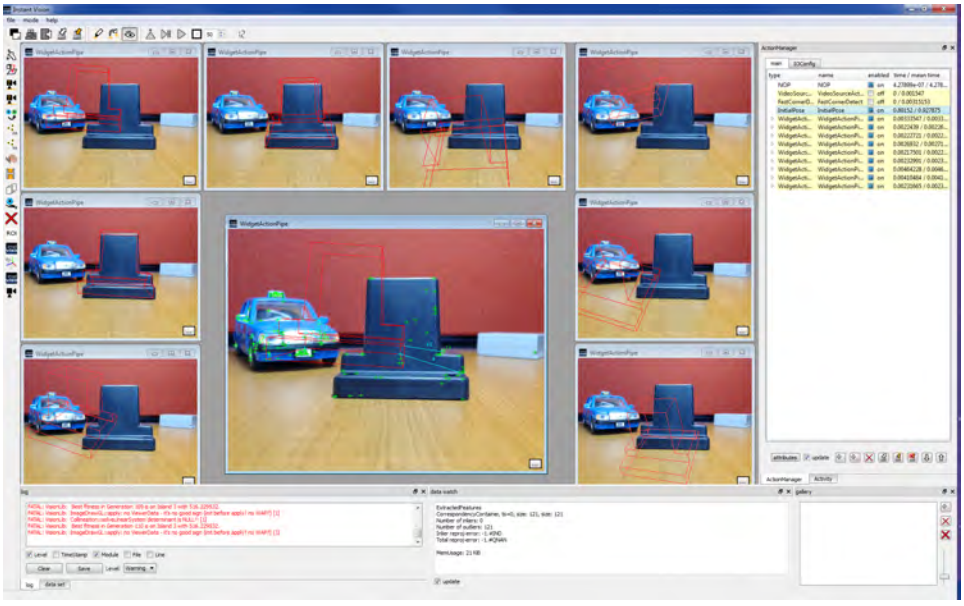

Abbildung 49: Hinzunahme eines detaillierten Objekts. Quelle: eigene Erstellung

7 Zusammenfassung

In der vorliegenden Bachelorarbeit wurde ein Algorithmus zur automatisierten Initialisierung des modellbasierten 3D-Tracking unter Verwendung eines *genetischen Algorithmus* entwickelt. Der Begriff der Automatisierung wurde dabei sehr weiträumig aufgefasst. Es wurde versucht, lediglich das Vorwissen anhand eines 3D-Feature Modells des zu trackenden Objekts und 2D-Feature aus einem Kamerabild zu verwenden.
Ein Ansatz mit nur so wenig Vorwissen ist neu. Die initiale Pose konnte sonst nur mit weitaus mehr Vorwissen bestimmt werden, was bedeutet, dass ein markerloses 3D-Tracking nicht ohne Weiteres möglich war.
Das System erhält als Eingabe eine Menge von 2D-Feature und 3D-Feature. Diese Feature können auf beliebige Weise detektiert worden sein. Anhand dieser Feature werden zufällig 2D-3D-Korrespondenzen zwischen den Mengen aufgebaut. Hat man insgesamt sechs solcher Korrespondenzen gewählt, wird eine Pose geschätzt. Mit ihr ist es möglich, jedes 3D-Feature einer Korrespondenz auf zweidimensionale Bildkoordinaten abzubilden. Indem für jede Korrespondenz der Pixelabstand zwischen 2D-Feature und abgebildetem 3D-Feature gemessen und aufsummiert wird, kann der Pose ein Fitnesswert zugewiesen werden.
Da dieser Ansatz einen sehr großen Suchraum mit sich bringt, nämlich den, wie viele Möglichkeiten es gibt, die Korrespondenzen herzustellen, war es notwendig den Suchraum optimiert zu durchlaufen. Dabei wurde das Optimierungsverfahren eines *genetischen Algorithmus* genutzt und durch eine Insel Evolution und einen eigenen Ansatz erweitert. Die Insel Evolution lässt im Grunde auf einer Anzahl Inseln jeweils einen *genetischen Algorithmus* laufen. Nach einer gewissen Anzahl von Generationen werden Individuen zwischen den Inseln ausgetauscht. Der eigene Ansatz nennt sich *Pest und Krankheit* und versucht das Dominanzproblem zu lösen, indem ebenfalls nach einer Anzahl von Generationen ohne Verbesserung des besten Individuums, die komplette Population, bis auf die Eliten ausgetauscht werden. Existieren keine Eliten, werden alle bis auf das beste Individuum ausgetauscht.
Zur Umsetzung dieses Konzepts war es notwendig einige Grundlagen zu erlernen. Darunter fiel die 2D-Feature Detektion in Bildern. Hier wurden die grundlegenden Verfahren wie der *Movarec Punktdetektor*, *Harris Punkt-*

detektor und der verwendete *FAST Punktdetektor* vorgestellt. Diese gewonnenen Feature wurden dann als 2D-Feature Menge an den Algorithmus übergeben. Neben den 2D-Feature sind 3D-Feature notwendig. Dazu wurde in Kapitel 2.2 der Begriff des *3D-Feature Modells* eingeführt. Dieses Modell beschreibt ein beliebiges Objekt mit 3D-Feature, die auf verschiedenste Art gewonnen werden können. Wichtig dabei ist nur, dass sie äquivalent zu den 2D-Feature sind. Das soll heißen, dass man nicht Punkte im 2D-Bild detektiert und Linien im 3D-Modell. Die verwendete Art des Feature muss gleich sein.
Die darauf folgenden Kapiteln beschäftigten sich mit der Frage, wie eine Pose aus Korrespondenzen geschätzt werden kann. Dazu wurden lineare und iterative Methoden vorgestellt. Da iterative Methoden eine initiale Pose benötigen, wurde eine lineare Methode gewählt. Aus Korrespondenzen, die zwischen den beiden Mengen aufgebaut werden, kann eine Pose geschätzt werden. Es ist aber zu beachten, dass diese in den meisten Fällen falsch sein wird. Der Suchraum, wie die Korrespondenzen gewählt werden können, ist, wie in Kapitel 4.2 erläutert, sehr groß. Um eine Pose zu bewerten wurde das Modell der *Mathematischen Projektion/Translation* in Kapitel 2.3 erläutert. Mit ihm lässt sich ein dreidimensionaler Punkt auf Bildkoordinaten abbilden. Damit ist es möglich, eine Korrespondenz zu bewerten.
Da es viele Möglichkeiten gibt, Korrespondenzen herzustellen, wurde zum Ende der Einleitung grundlegende Optimierungsverfahren vorgestellt. Darunter waren *Partikelfilter*, *Simulated Anealing* und *genetischer Algorithmus*.
Nachdem die Grundlagen zur Realisierung der Idee vermittelt wurden, ging es weiter mit dem Stand der Technik. Nach eingehender Literaturrecherche hat sich gezeigt, dass dieses Problem entweder unter Zuhilfenahme von weiterer Technik, wie GPS- und Kompassdaten und/oder durch Benutzerinteraktion, wie das Bewegen der Kamera oder händischen Anpassung des 3D-Modells, geschieht. Außerdem wurde in diesem Kapitel auf den aktuellen Forschungsstand der *genetischen Algorithmen* eingegangen. Dabei hat sich gezeigt, dass die sogenannte Insel Evolution eine beliebte Erweiterung ist. Durch sie wird sich ein effizienteres Durchsuchen des Suchraums und Vermeidung des Dominanzproblems erhofft.
An dieses Kapitel schließt sich das Konzept an. Dies wurde bereits einleitend zu diesem Kapitel erläutert und wird hier übersprungen. Darauf folgt das Kapitel 5 die Realisierung. Hier wurde verwendete Software genannt, die zum Umsetzung der Problemstellung nützlich war und auch Angaben zur Implementierung wurden gemacht. Da der Algorithmus in die *Instant Vision* eingebunden werden soll, konnte zudem auf einige Implementationen, wie dem *FAST Punktdetektor* und der linearen Posenschätzung zurückgegriffen werden.

7.1 Fazit

Die durchgeführten Tests in Kapitel 6 haben gezeigt, dass der Ansatz erfolgreich ist. Für das Testszenario konnte eine Trefferquote von etwa 90% bei einer Laufzeit von ungefähr zwölf Minuten erzielt werden.
Allerdings konnte der Begriff der Automatisierung nicht ganz so weiträumig aufgefasst werden wie gewünscht. Es ist notwendig, die grobe Entfernung des Objekts anzugeben. Ansonsten wird das 3D-Feature Modell auf wenige oder nur einen 2D-Feature abgebildet, was eine sehr gute Fitness bedeutet, aber im Kontext dieser Arbeit keine gültige Pose ist. Es lässt sich festhalten, dass dieser Ansatz am besten für größere Objekte funktioniert. Das Objekt braucht nicht bildfüllend zu sein, aber groß genug, um ausreichend 2D-Feature zu liefern. Je nach Anwendungsfall muss der Parameter *Translation Z* angepasst werden. Wie die Größe des Objekts im Kamerabild sich exakt auf den Algorithmus auswirkt, konnte nicht genau evaluiert werden. Dazu wurden nur zwei Versuche gemacht, die zeigten, dass der Algorithmus mit einem Objekt, das nahezu bildfüllend und einem, das lediglich 5% des Bildes ausmacht, zurecht kommt. Um eine repräsentative Aussage treffen zu können, muss dies allerdings genauer evaluiert werden.

7.2 Ausblick

Wie in Kapitel 3.3.2 erwähnt, wurde kürzlich ein Ansatz vorgestellt, der das Verfahren des *genetischen Algorithmus* auf einer Grafikkarte parallelisiert. Mit der angegebenen Beschleunigung gegenüber einer CPU-Lösung wäre es möglich die Laufzeit des Algorithmus auf wenige Sekunden zu senken. Inwieweit die vorgeschlagene GPGPU-Implementierung auch auf den *genetischen Algorithmus* dieser Arbeit passt, vielmehr die Umsetzung der Fitnessfunktion, die das Herzstück des Konzepts darstellt, kann zu diesem Zeitpunkt nicht abgeschätzt werden. Im Allgemeinen stellt die Laufzeit des Algorithmus noch ein grundlegendes Problem zur Verwendung dar. Dies wird sich zum einen durch schnellere Computer in den nächsten Jahren automatisch lösen und zum anderen wäre es sinnvoll, die Auswahl der Korrespondenzen intelligenter vorzunehmen. Dies könnte durch die Annahme geschehen, dass die meisten 2D-Feature von dem gesuchten Objekt kommen. Denn der Benutzer wird sehr wahrscheinlich die Kamera auf dieses Objekt ausrichten, was auch bedeutet, dass dieses Objekt durch den Autofokus[27] am schärfsten dargestellt wird. Das wiederum zur Folge hat, dass es in dem Bereich des scharfgestellten Objekts am meisten 2D-Feature gibt. Hat man eine Region mit einem verstärktem Vorkommen von 2D-Feature identifiziert, kann man diese zur Korrespondenzbildung bevor-

[27] *Wikipedia (26.04.2013):* "Als Autofokus (AF) wird die Technik einer Kamera oder allgemein eines jeden optischen Apparates bezeichnet, automatisch auf das Motiv scharfzustellen."

zugen.
Ein anderer Ansatz wäre die Einschränkung zu verlangen, dass das gesuchte Objekt im Bildzentrum zu positionieren ist. Damit könnte man die 2D-Feature des Bildzentrums bevorzugt nutzen. Mit dieser Art der Einschränkung würde aber der Ansatz in den Bereich der *Benutzergestützten Posenbestimmung* abwandern.
Ein weiterer Verbesserungsvorschlag wäre die Nutzung der Linieninformation des 3D-Modells. Damit könnte die Fitnessfunktion um einen Linienvergleich zwischen 3D-Modell und Kamerabild erweitert werden. Dafür müsste zunächst das Kamerabild mit einem Liniendetektor, wie beispielsweise dem *Canny Edgedetektor*[61], gefiltert werden. Dieses Verfahren setzt allerdings voraus, dass ein 3D-Linienmodell zur Verfügung steht. Ist dies nicht der Fall, müsste auch das 3D-Modell mit einem Liniendetektor gefiltert werden. Dazu sei auf [44] verwiesen. Mit dieser Idee kann versucht werden, die Angabe des *Translation Z*-Parameter zu entfernen oder großzügiger zu gestalten. Es ist zudem möglich, eine zweite Aussage über die Qualität der Pose zu treffen, die nicht direkt von den Korrespondenzen abhängig ist.
Ein weiterer wichtiger Punkt für zukünftige Arbeiten ist, zunächst ein anderes Objekt zu tracken und auch den Algorithmus mit einer anderen Art von Feature zu evaluieren.

8 Anhang

8.1 Symbolverzeichnis

Symbol	Bezeichner
A	Autokorrelationsmatrix eines Punktes p
$A_{m,n}$	Autokorrelationsmatrix für eine Punktregion (m, n)
(m, n)	Eine Punktregion
$H_{m,n}$	Ist der Harris-Operator in der Region (m, n)
$V_{i,j}$	Minimum von $V^1_{i,j}$, $V^2_{i,j}$, $V^3_{i,j}$, $V^4_{i,j}$
$V^i_{i,j}$	Summe der quad. Diff. zw. Pixeln in einer Hauptrichtung
I_x	Ableitung in x-Richtung
I_y	Ableitung in y-Richtung
$K^{(x)}$	Ableitungskern in x-Richtung
$K^{(y)}$	Ableitungskern in y-Richtung
p	Bildpunkt an einer Stelle (x, y)
p	Ein Punkt in Bildkoordinaten
$loc(p)$	Pixelort von p
$val(p)$	Pixelwert von p
K_i	intrinsische Kameraparameter, auch Kameraintrinsik
K_e	extrinsische Kameraparameter, auch Kameraextrinsik
Ψ	Gesamtdistanz von K_e
$d(p_1, p_2)$	Euklid′scher Abstand zwischen zwei Punkten
Wsk_i	Wahrscheinlichkeit des i-ten Individuums
$UWsk_i$	Umgekehrte Wahrscheinlichkeit des i-ten Individuums

8.2 Listings

```
void geneticAlgorithmSimple(  double maxFitness,
                              int maxGeneration,
                              int popSize)
{
  int actGeneration = 0;
  Population* pop   = createPopulation(popSize);

  pop->evaluate();

  double fitness    = pop->getBestIndividual()->getFitness();

  while(fitness < maxFitness && actGeneration < maxGeneration)
  {
    pop->selection();
    pop->crossover();
    pop->mutation();
    pop->evaluate();

    fitness = pop->getBestIndividual()->getFitness();

    actGeneration++;
  }
}
```

Listing 3: Pseudocode genetischer Algorithmus

```
#VRML V2.0 utf8

Shape {
  appearance Appearance {
    material Material {
      # note emissive color makes these dots easier to see
      emissiveColor 1 0 0
    } # end material
  } # end appearance
  geometry IndexedLineSet {
    coord Coordinate {
      point [
      #Look from the side
        #base of lampstand
        -2.5 -2.2  2.5,     # point 0, left and close
        -2.5 -2.2 -2.5,     # point 1, left and back
         2.5 -2.2 -2.5,     # point 2, right and back
         2.5 -2.2  2.5,     # point 3, right and close

        #top base of lampstand
        -2.5 -1.4  2.5,     # point 4, left and close
        -2.5 -1.4  1.3,     # point 5, left and middle near
        -2.5 -1.4  -1.3,    # point 6, left and middle far
        -2.5 -1.4 -2.5,     # point 7, left and back
         2.5 -1.4 -2.5,     # point 8, right and back
         2.5 -1.4  -1.3,    # point 9, right and middle far
         2.5 -1.4  1.3,     # point 10, right and middle near
         2.5 -1.4  2.5,     # point 11, right and close


         #top first storey of lampstand
```

```
32          -2.5 -0.7  1.3,     # point 12, left and close
33          -2.5 -0.7 -1.3,     # point 13, left and back
34          -1.55 -0.7 -1.3,    # point 14, middle left and back
35           1.55 -0.7 -1.3,    # point 15, middle right and back
36           2.5 -0.7 -1.3,     # point 16, right and back
37           2.5 -0.7  1.3,     # point 17, right and close
38           1.55 -0.7 1.3,     # point 18, middle right and front
39          -1.55 -0.7 1.3,     # point 19, middle left and front


42           #top second storey of lampstand
43          -1.4 2.355  1.05,   # point 20, left and close
44          -1.4 2.355 -1.05,   # point 21, left and back
45           1.4 2.355 -1.05,   # point 22, right and back
46           1.4 2.355  1.05,   # point 23, right and close
47        ]  # end point
48      } # end coord

50      coordIndex [
51        #base
52        0, 1, 2, 3, 0, -1,
53        4, 5, 10, 11, 4, -1,
54        6, 7, 8, 9, 6, -1,

56        #first storey
57        12, 13, 14, 19, 12, -1,
58        15, 16, 17, 18, 15, -1,

60        #top
61        20, 21, 22, 23, 20, -1,

63       #vertical edges
64        #First Cube
65        0, 4, -1,
66        1, 7, -1,
67        2, 8, -1,
68        3, 11, -1,

70        #Second Cube
71        5, 12, -1,
72        6, 13, -1,
73        9, 16, -1,
74        10, 17, -1,

76        #Third Cubde
77      19, 20, -1,
78        14, 21, -1,
79        15, 22, -1,
80        18, 23, -1,


83      ] # end coordIndex


86    } # end geometry
87  } # end shape
```

Listing 4: Beispiel der *VRML* Datei des Lampenständers

Abbildungsverzeichnis

Sachregister

Literatur

[1] E. Truco and A. Verri, *Introductory Techniques for 3-D Computer Vision*. Upper Saddle River, New Jersey, 1998.

[2] M. Dennhardt, "Kamerapositionsbestimmung über Analyse durch Synthese," 2007.

[3] S. Achilles, "Markerloses Tracking unter Verwendung von Analyse durch Synthese auf Basis von Featuredetektoren," 2008.

[4] H. P. Morevec, "Towards automatic visual obstacle avoidance," pp. 584 – 584, 1977.

[5] C. Harris and M. Stephens, "A combined corner and edge detector," *Alvey vision conference*, pp. 147–152, 1988.

[6] L. Priese, "Skript zu den Vorlesungen Bildverarbeitung 1 und 2 Verarbeitung und Analyse digitaler Bilder Vorwort," *Universitaet Koblenz-Landau, Vorlesungsfolien*, pp. 300+, 2012.

[7] U. A. Lienhart, Rainer (Multimedia Computing, "The harris corner detector," *Universitaet Augsburg, Vorlesungsfolien*, no. April, 2008.

[8] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *Computer Vision ECCV 2006*, 2006.

[9] S. M. Smith and J. M. Brady, "SUSAN A New Approach to Low Level Image Processing 2 The SUSAN Principle for Feature Detection," pp. 1–59, 1995.

[10] J. E. Bresenham, *Algorithm for computer control of a digital plotter*. IBM Systems Journal 4, 1965.

[11] E. Rosten, "High performance rigid body tracking University of Cambridge," no. February, 2006.

[12] E. Rosten, G. Reitmayr, and T. Drummond, "Real-time video annotations for augmented reality," *Advances in Visual Computing*, 2005.

[13] T. M. Mitchell, "No Title," in *Machine Learning*, pp. 55–58, 1997.

[14] H. Wuest, "Efficient Line and Patch Feature Characterization and Management for Real-time Camera Tracking," *d-nb.info*, 2008.

[15] A. Hartley, R. I. and Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second ed., 2004.

[16] C. Gaida, "Untersuchung von Verfahren zur Pose-Schätzung im Hinblick auf Analyse durch Synthese," *uni-koblenz.de*, 2011.

[17] D. Dementhon and L. Davis, "Model-based object pose in 25 lines of code," *International Journal of Computer Vision*, pp. 123 – 141, 1995.

[18] L. Oberkampf, D. and DeMenthon, D.F. and Davis, *Iterative pose estimation using coplanar points*. 1993.

[19] D. G. Lowe, *Fitting parameterized three-dimensional models to images*. 1991.

[20] E. Lu, C.-P., Hager, G. D., Mjolsness, "Fast and globally convergent pose estimation from video images," pp. 610 – 622, 2000.

[21] W. Alt, *Nichtlineare Optimierung*. Vieweg Verlag, 2002.

[22] J. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, 1965.

[23] W. C. Davidon, *Variable Metric Method for Minimization*. 1991.

[24] D. Marquardt, *An Algorithm for Least-Squares Estimation of Nonlinear Parameters*. 1963.

[25] A. H. D. del Pino, "Vorlesung "Genetische Algorithmen"Foliensatz 1 SS2009," 2009.

[26] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39–43.

[27] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, pp. 1942–1948, 1995.

[28] S. Wagner and T. U. Chemnitz, "Partikelschwarmoptimierung," 2010.

[29] H. Holland H., *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. 1992.

[30] H. T. U. W. Feltl, "Ein Genetischer Algorithmus für das Generalized Assignment Problem," no. April, 2003.

[31] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[32] D. Cohoon, J., Hegde, S., Martin, W., Richards, "Punctuated equilibria: a parallel genetic algorithm," *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 148–154, 1987.

[33] D. Whitley, S. Rana, and R. Heckendorn, "The island model genetic algorithm: On separability, population size and convergence," pp. 1–17, 1999.

[34] Http://www.instantreality.org, "Projektseite Instant Vision."

[35] M. Wimmer and B. Radig, "Initial pose estimation for 3D model tracking using learned objective functions," *Computer Vision ACCV 2007*, 2007.

[36] M. Wimmer and B. Radig, "Automatically Learning the Objective Function for Model Fitting," *ias.cs.tum.edu*, 2007.

[37] S. Ekvall, D. Kragic, and F. Hoffmann, "Object recognition and pose estimation using color cooccurrence histograms and geometric modeling," *Image and Vision Computing*, vol. 23, pp. 943–955, Oct. 2005.

[38] T. Kobayashi and R. Kddi, "Robust 2D-3D Matching for 3D Object Pose Detection," vol. 1, no. 212, pp. 3–6, 2012.

[39] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Real-time detection and tracking for augmented reality on mobile phones.," *IEEE transactions on visualization and computer graphics*, vol. 16, no. 3, pp. 355–68, 2010.

[40] D. Stricker, "Computer-Vision-basierte Tracking-und Kalibrierungsverfahren für Augmented Reality," 2003.

[41] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection," *... Analysis and Machine ...*, pp. 1–35, 2010.

[42] J. Rubio, M. Hidalgo, and F. López, "A new method for detection and initial pose estimation based on Mumford-Shah segmentation functional," *Pattern Recognition and Image Analysis*, 2003.

[43] D. Mumford and J. Shah, "Optimal approximations by piecewise smooth functions and associated variational problems," *Communications on pure and applied ...*, vol. XLII, 2006.

[44] S. Kowalczyk, "Initialisierung im markerlosen Tracking mit dem Analyse-durch-Synthese Ansatz in einem urbanen Kontext," 2012.

[45] G. Bleser and D. Stricker, "Advanced tracking through efficient image processing and visual inertial sensor fusion," *Computers & Graphics*, pp. 137–144, 2008.

[46] T. Habelitz, "Markerloses Tracking unter Verwendung von Analyse durch Synthese auf Basis der Ähnlichkeitsbestimmung photorealistischer Bilder," 2009.

[47] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov. 2004.

[48] H. Wuest, D. Stricker, and J. Herder, "Tracking of industrial objects by using CAD models," *Journal of Virtual Reality and Broadcasting*, vol. 4, no. 1, 2007.

[49] Http://www.metaio.com, "Metaio Homepage."

[50] Http://www.youtube.com/watch?feature=player_embedded&v=xw3M-TNOo44, "Metaio Outdoor Tracking."

[51] W. Lienig and J. Cohoon, "C6. 3 Parallel Genetic Algorithms Based on Punctuated Equilibria," *Citeseer*, pp. 1–15, 1997.

[52] C. Pettey, M. Leuze, and J. Grefenstette, "A parallel genetic algorithm," 1987.

[53] R. Tanese, "Parallel genetic algorithms for a hypercube," pp. 177–183, 1987.

[54] R. Tanese, "Distributed genetic algorithms," pp. 434–439, 1989.

[55] T. Belding, "The distributed genetic algorithm revisited," pp. 114–121, 1995.

[56] P. Pospichal and J. Jaros, "GPU-Based Acceleration of the Genetic Algorithm," pp. 2–3, 2010.

[57] R. Shah, P. Narayanan, and K. Kothapalli, "GPU-Accelerated Genetic Algorithms," *cvit. iiit. ac. in*.

[58] A. H. D. del Pino, "Genetische Algorithmen 3. Vorlesung," 2009.

[59] B. P. . M. Selzam, "Genetische Algorithmen,"

[60] W. Martin, J. Lienig, and J. Cohoon, "Island (migration) models: evolutionary algorithms based on punctuated equilibria," *...of evolutionary ...*, pp. 1–16, 1997.

[61] J. Canny, "A computational approach to edge detection.," *IEEE transactions on pattern analysis and machine intelligence*, vol. 8, pp. 679–98, June 1986.