Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Echtzeit-Objekt-Verfolgung mit aktiven Markern und Multi-Kamera-System

Bachelorarbeit

vorgelegt am: 16. April 2014

am Fachbereich Informatik der Universität Hamburg

Name:	Johannes Schlundt
Matrikelnummer:	6247346
Arbeitsbereich:	Technical Aspects of Multimodal Systems (TAMS)
Studiengang:	Informatik
Studienjahrgang:	2010
Erstgutachter:	Dr. Norman Hendrich
Zweitgutachter:	Eugen Richter

Zusammenfassung

Objekt-Verfolgung durch ein Computer-System ist ein Bereich mit vielen Möglichkeiten. Eine Variante davon ist mit codierten Markern an Objekten, welche die Erfassung mit Kameras und die Zuordnung für den Computer und deren Software sehr erleichtern kann. Es entfallen entsprechende Kenntnisse zu dem Aufbau des Objektes, da die Marker genau aussagen können, was zu sehen ist.
Diese Arbeit beschäftigt sich mit der Entwicklung von aktiven Markern, deren Kennzeichnung und der Möglichkeit einer Echtzeit-Verfolgung. Dabei befinden sich die aktiven Marker auf einem Entwickler-Board, mehrere Kameras erfassen dies und stellen diese in einem 3D-Raum dar. Das Ganze geschieht mit handelsüblicher Hardware und freien Software. Die aktiven Marker durchlaufen dabei Periodisch eine feste Sequenz aus dunklen und hellen Phasen, die das dazugehörige Programm erfassen und zuordnen kann. Durch die Zuordnung erfolgt die 3D-Rekonstruktion in wenigen schritten, da das suchen der passenden Paaren entfällt.

Abstract

Object tracking by a computer system is a topic with many possibilities. One option are active markers on objects that allow detection and recognition with cameras and the mapping for the computer system and their software. It omitted relevant knowledge about the structure of the object, so the markers can say exactly what it is.
This work deals with the development of active markers, their identification and the possibility of real-time tracking. The active markers are on a development board, multiple cameras capture and show them in a 3D-graphics, all this done with off-the-shelf hardware and free for use software. The active marker run a fixed periodic sequence of dark and light phases, which can detect and assign the associated program. By mapping the 3D reconstruction it need to be performed just few steps since search the matching pairs be eliminated.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Es gibt viele Wege, menschliche Bewegungen mit Kamera-Systemen zu erfassen. Diese Arbeit zeigt einen Fall, den jeder mit entsprechender Hardware durchführen kann. Für die Erfassung wird meistens das Wissen des zu erfassenden Objekts mit einbezogen, die als Modelle gewertet werden können. Diese Modelle werden genutzt, um das erfasste Objekt korrekt zu rekonstruieren und darzustellen. Das Zurechtrücken des Modells zum erfassten Objekt kostet dabei die meiste Zeit und ist deshalb meistens nur mit Nachbearbeitung möglich oder mit Hardware, die mit erheblichen Kosten verbunden ist. Das Ziel in dieser Bachelorarbeit ist aktive Marker zu entwickeln und somit eine Alternative oder Erweiterung zu der passiven Methode zum HANDLE [Ric11] Projekt zu bieten. Die Marker werden von mehreren Kameras identifiziert und zugeordnet. Die Kameras befinden sich dabei auf einem Gerüst und ermöglichen durch die verschiedenen Ansichten eine spätere 3D-Rekonstruktion. Durch die Identifizierung im Bild wird der Rechenaufwand für die 3D-Rekonstruktion verringert und somit zur Echtzeitverarbeitung beitragen.
Die Identifizierung, welcher Marker im Bild zu sehen ist, geschieht durch die einzigartige Sequenz von hellen und dunklen Phasen, die jeder Marker besitzt. Jede Kamera hat eine andere Sicht zu den Markern und durch Kombination der Blickwinkel und das Wissen, welcher Marker im Bild zu sehen ist, ergibt sich am Ende eine dreidimensionale Darstellung.
Durch diese Erkennungsmöglichkeit können wir menschliche Bewegungen und/oder Handbewegungen von einem Computer leichter erfassen, verarbeiten, abspeichern und z.B. auf einem Roboterarm umsetzen. In der Diplomarbeit (vgl. [Ric11, Seite 62ff]) wird eine Hand rekonstruiert, ohne das Wissen dazu, was der Computer genau von der Hand erfasst. Mit der Möglichkeit mit aktiven Marker, die in dieser Arbeit beschriebenen wird, können wir direkt die Hand[1] oder Teile von der Hand umsetzen, da wir für jeden Marker einen Namen oder Kennzeichnung vergeben können und das Zuordnen für das Programm somit erleichtern.

[1] Da das Rekonstruieren einer Hand nicht so leicht ist, wird in dieser Arbeit nur das Erfassen der aktiven Marker beschrieben, da für die korrekte Darstellung einer Hand noch mathematische Korrekturen benötigt werden.

1.2 Genutzte Software und Hardware

Das in dieser Arbeit beschreibende System wird auf einem Standard-PC ausgeführt. Dabei dienen drei preiswerte Kameras als optische Sensoren. Die aktiven Marker sind einfache LEDs auf einem Arduino Entwickler-Board, der die verschiedenen Sequenzen der LEDs modelliert.
Als Grundlage diente die Diplomarbeit von Eugen Richter [Ric11]. Aus dieser Arbeit wurden nur wenige Elemente, wie etwa die Kameraansteuerung mithilfe von V4L2 API und die 3D-Rekonstruktion (ohne Hand Rekonstruktion) übernommen und zum größten Teil auf die aktuelle Version der OpenCV Bibliotheken angepasst.
Der Großteil des in dieser Arbeit dokumentierten Funktionsumfang wurde selbstständig entwickelt und umfasst die Erfassung der aktiven LED Marker auf einem Arduino Due Board, die Sequenz Erkennung, dessen Entschlüsselung und die Anpassung der 3D-Rekonstruktion an das neue Programm.

- Für die Entwicklung wurde der Qt Creator als Entwicklungsumgebung (Version 3.0) mit Qt 4.8 als Programmbibliothek [Pro13] genutzt.
- Die Bildverarbeitung wurde mit der OpenCV 2.4.7 Programmbibliothek [Ope13c] realisiert.
- Als Betriebssystem wurde Ubuntu 12.10 mit einem Quad-Core Prozessor und 8 GB Arbeitsspeicher verwendet.
- Arduino Due [Ard14a] wurde als Entwickler-Board, mit der Entwicklungsumgebung von Arduino (Version 1.5.5) und der zusätzlichen DueTimer [Gom13] Programmbibliothek verwendet.
- Die Kameras sind mehrere PlayStation®Eye's [Son13] mit einer Auflösung von 640 x 480 Pixeln und einer Bildfrequenz von 60 Hz.
- Die Kamerakalibrierung wurde mit der „Camera Calibration Toolbox for Matlab®“ [Bou13] bewerkstelligt.

1.3 Verwandte Arbeiten

Es existieren mittlerweile viele Wege, mit einer Kamera ein Objekt zu erfassen und dies dann am Computer zu rekonstruieren. Die Echtzeitfähigkeit wird dabei meist im Hintergrund gelassen. So gibt es die Möglichkeit, mit passiven Markern dem Computer zu helfen, die Objekte zu erfassen. Den passiven

Abbildung 1: Passive Methode, die heutzutage kommerziell genutzt wird. Quelle: [Wik13d]

Marker gibt es z.B. mit einer reflektierenden Oberfläche, die von Kameras erfasst werden, die empfindlich auf infrarotes Licht sind. Dabei wird das infrarote Licht in der direkten Nähe der Kameras ausgestrahlt. Die passiven Marker reflektieren dabei das infrarote Licht direkt zum Ursprung, sodass die Kameras die Marker leichter erfassen können [Wik13d]. In der Abbildung 1 sieht man an der Person die Marker, die hell erscheinen. An der Decke befinden sich mehrere Kameras, die die Marker erkennen und diese Informationen an einen Computer senden und wodurch ein 3D-Modell rekonstruiert wird. Diese Möglichkeit wird heutzutage in Filmen, Computerspielen und ähnlichem genutzt. Es gibt auch die Alternative mit verschiedenen farbigen Markern eine

Hand zu erfassen, wie es in der Diplomarbeit von Eugen Richter [Ric11] gezeigt wird. Dabei geht es um farbige passive Marker, die an markanten Stellen der Hand befestigt sind und aufwendig am Computer rekonstruiert und dargestellt werden. Als Erstes werden fünf Farben mit einer Kamera erfasst und

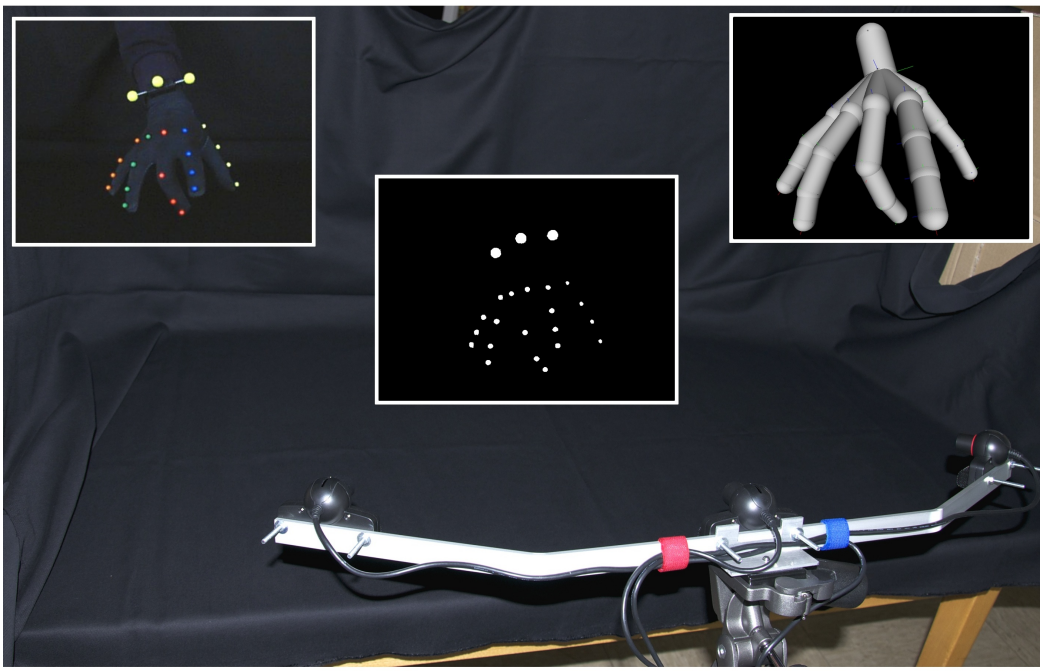


Abbildung 2: Allgemeiner Entwurf vom Ansatz (von links nach rechts): 1. Bewegungsaufnahme mit den drei-Kamera Einstellungen - 2. Bildverarbeitung und Stereo Rekonstruktion - 3. Rekonstruktion und Darstellung der Hand. Quelle: [Ric11, Seite 4]

kalibriert. Die fünf Farben sind dieselben Farben der passiven Marker und diese Farben stellen die fünf einzelnen Finger da. Die Kalibrierung der Farben dient dazu, die farbigen Marker später in den Bildern segmentieren und extrahieren zu können. Nach der Kalibrierung zieht man sich einen schwarzen Handschuh mit den farbigen Markern an, die sich an den markanten Stellen befinden. Danach hält man die Hand vor einen schwarzen Hintergrund und führt die Bewegungen aus, die aufgenommen und verarbeitet werden sollen. Die Kameras befinden sich in verschiedenen Positionen, um dreidimensional die Marker zu erfassen. Am Ende werden dann die Bilder, Bild für Bild von der Festplatte verarbeitet, angezeigt und die Marker extrahiert. Die extrahierten Marker, aus verschiedenen Blickwinkeln betrachtet, werden in ein 3D-Koordinatensystem umgerechnet. Die 3D-Koordinaten werden dann mit

einem zuvor definierten Handmodell verglichen und die Hand rekonstruiert. Am Ende erhält man die erfasste Hand als eine 3D-Rekonstruktion in einem Fenster dargestellt. Der Ablauf wird in Abbildung 2 veranschaulicht.

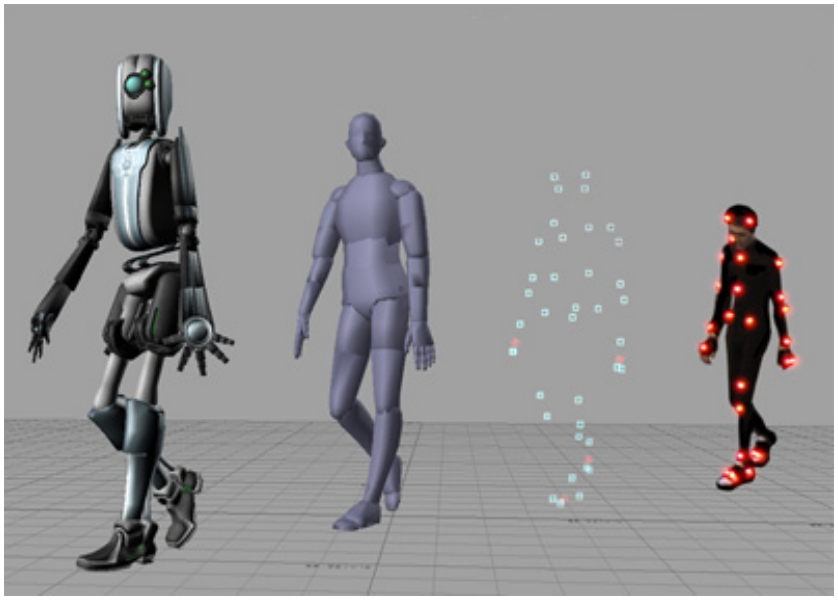
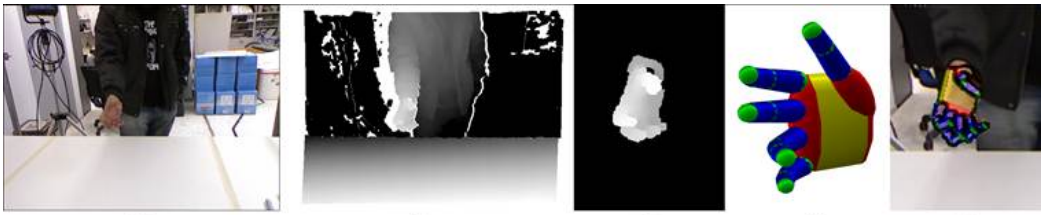


Abbildung 3: Überführung von einem erfassten Menschen zu einer fiktiven Figur. Quelle: [Wik13e]

Eine weitere Möglichkeit ist die Nutzung von aktiven Markern, anstelle von passiven Markern. Dies ermöglicht dann über die Zeit mehr Informationen in einem Punkt zu bringen [Wik13e]. Diese Möglichkeit nutzt das Unternehmen „PhaseSpace“ mit seinen „IMPULSE X2“ [Pha12a] System. Dabei ist es möglich 2 bis 48 Kameras an einem System zu verwenden. Wenn man mehrere Systeme kombiniert, können auch mehr Kameras angeschlossen werden. Die aktiven Marker werden von den Kameras mit 12 MP und 960 Hz erfasst und haben dank den Onboard Prozessor eine Subpixelauflösung von bis zu 36.000 x 36.000 Pixeln.
Jeder aktive Marker besitzt dabei sein einzigartiges 8-Bit Muster, dies erlaubt in Echtzeit bis zu 512 unterscheidbare Marker zu verfolgen [Pha12b]. Der LED-Controller kann dabei bis zu 72 LEDs ansteuern und direkt von dem Server beeinflusst werden. Die Kameras sind über ein Gigabit Netzwerk mit dem Server verbunden. Dieser ganze Aufbau ermöglicht bis zu 8 Menschen, je Server, in Echtzeit zu verfolgen. Die erfassten Daten können dann an ein Animationsprogramm übertragen und abgespielt werden.

Ein anderer Ansatz ist die markerlose Variante. Dabei wird aus einer Tiefeninformation gleich auf eine 3D-Struktur geschlossen. In den vorhergehenden Beschreibungen wurde aus mehreren Kameras in verschiedenen Positionen eine 3D-Struktur errechnet. In der Arbeit mit dem Titel „Efficient Model-based 3D Tracking of Hand Articulations using Kinect“ [OKA11] wird die Möglichkeit der Verfolgung einer Hand mithilfe des Kinect-Sensors gezeigt. Dabei wird die Tiefeninformation genutzt, welche die Kinect-Kamera liefert, um ein vordefiniertes Handmodell entsprechend auf die erfasste Hand abzubilden, auszurichten und darzustellen. Zur Erhöhung der Genauigkeit und zur Optimierung des Problems wird eine Variante von „Particle Swarm Optimization“ genutzt, um Fehler in der Erkennung auszubessern. Je nach Hardware und GPU-Leistung kann eine Echtzeit Verfolgung mit einer Frequenz von etwa 15 Hz erreicht werden.



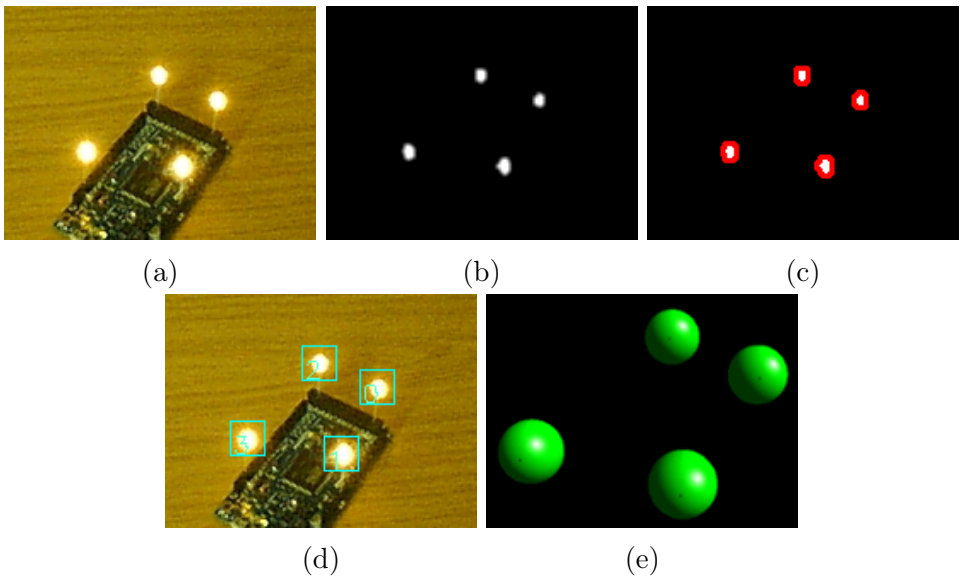
Graphische Darstellung der Methode. Ein Kinect RGB Bild (a) und die entsprechende Tiefenabbildung (b). Die Hand ist segmentiert (c) durch die gemeinsame Berücksichtigung der Hautfarbe und Tiefe. Die Methode passt das Handmodell (d) mit den rekonstruierenden Handgelenk (e) in der Abbildung an. Quelle: [Arg13]

1.4 Aufbau dieser Arbeit

Diese Arbeit beginnt mit grundlegenden Informationen zum Programm. Als Erstes wird die Hardware erklärt, später wird die Software beschrieben. Danach wird die Programmoberfläche und die Kamerakalibrierung beschrieben. Erst dann wird die Abfolge der Erkennung genauer beschrieben. Zum Schluss werden Ergebnisse einer realen Erfassung gezeigt und Vorschläge einiger Optimierungsmöglichkeiten des Programms dargestellt.

1.5 Idee und Vorstellung

Die Idee dieser Arbeit ist es mithilfe von aktiven Markern die erkannten Marker im Kamerabild zu identifizieren, sie zu kennzeichnen und am Ende in einem Fenster darzustellen. In Abbildung 4 sieht man die Verarbeitungsschritte für die Erkennung. Die aktiven Marker sind mehrere LEDs an einem Arduino Due Board [Ard14a]. Das Arduino-Board erzeugt dazu für jede LED jeweils ein einzigartiges 16-Bit-Muster, die mit hellen und dunklen Phasen dargestellt werden. Die Möglichkeit mit hellen und dunklen Phasen die einzigartigen Sequenzen zu erfassen und zu verarbeiten, soll in Echtzeit ablaufen und gleichzeitig mit mehreren Kamera-Systemen funktionieren, sodass eine räumliche Erfassung erfolgen kann. Nach der Erfassung und Erkennung der Sequenz soll eine 3D-Rekonstruktion folgen und die erfassten Marker darstellen.



(a) (b) (c)

(d) (e)

Abbildung 4: Die jeweiligen Verarbeitungsschritte. Die Bilder sind vergrößerte Ausschnitte, um die Marker besser zu sehen. (a) Originalbild. (b) Entfernung der Farbinformationen und Umwandlung in ein Binärbild. (c) Das Resultat der OpenCV Kontursuche. (d) Originalbild mit Einzeichnung der zuständigen „Worker“. (e) Resultat der 3D-Rekonstruktion.

2 Drei-Kamera-System

Die Erfassung der aktiven Marker geschieht durch einfache Kameras. Dabei werden mehr als nur eine Kamera genutzt, um den Erfassungsbereich zu erweitern und eine 3D-Rekonstruktion zu ermöglichen.
Das Multi-Kamera-System ist, wie in Abbildung 5 zu sehen, auf einem gebogen Rahmen befestigt. Auf dem Rahmen wurden drei PlayStation®Eye Kameras befestigt, die jeweils zu der mittleren Kamera mit einem Winkel von etwa 30° stehen.
Die Kameras nutzen dabei eine Objektiv-Einstellung für ein Sichtfeld von 75°.

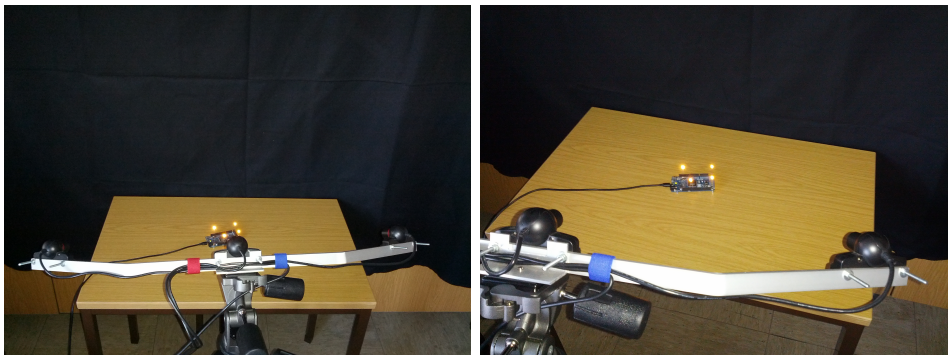


Abbildung 5: Versuchsaufbau mit drei Kameras und einen Arduino Due, mit vier LEDs als aktive Marker.

Damit die Verfolgung der LEDs leichter für die Software ist, werden alle automatischen Funktionen, wie etwa Weißabgleich, der Kamera deaktiviert und das Kamerabild so dunkel gesetzt wie möglich. Durch das Abdunkeln, sieht man die LEDs besser, und durch die Abschaltung der automatischen Funktionen werden mögliche Störungen in der Sequenzerkennung verhindert.

2.1 Kamerabild

Ein Bild wird in diesem Programm als ein zweidimensionales Array aus dem Wertebereich von 0 bis 255 dargestellt. Da die erfassten Bilder farbig sind, sind es 3 Werte von 0-255 für jeweils einen Farbkanal, dass hier aus einem BGR[2]-Farbraum besteht. Die Farbe entsteht beim Mischen der Grundfarben, wenn auf jeden Farbkanal ein Pixel jeweils 0 je Farbkanal ist, dann steht es für die Farbe schwarz. Wenn beim Farbkanal Rot der Wert 255 steht, bei Blau und Grün jeweils 0, dann sieht man die Farbe Rot. Wenn alle Farbkanäle 255 haben, dann ist Weiß zu sehen (siehe Abbildung 6).

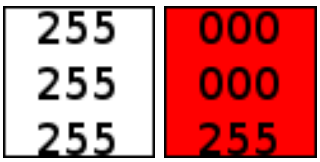


Abbildung 6: Die Zahlen zeigen den Wert für ein Pixel an. Von oben nach unten sind es die einzelnen B-,G- und R-Werte.

[2]BGR steht hier für Blau, Grün und Rot.

3 Arduino

Die Sequenzen (mehr ab Seite 15) der einzelnen LEDs werden vorher fest in den Arduino-Microcontroller einprogrammiert. Dieselbe Sequenz muss im PC-Programm mit der entsprechenden Kennzeichnung vorhanden sein. Eine minimale Helligkeit kann eingestellt werden, um es für das PC-Programm einfacher zu gestalten die LEDs bei einer dunklen Phase der Sequenz zu verfolgen.

3.1 Pulsweitenmodulation

Der Code für minimale Helligkeit wurde im Rahmen dieser Arbeit entworfen. Als Vorlage diente die Information zur Pulsweitenmodulation von Arduino-Webseite (vgl. [Hir13] und dessen Abbildung 7).
Dabei wurde versucht das Gleiche zu modellieren, was in Abbildung 7 gezeigt ist. Dazu wurde ein Zähler erstellt, der nach jedem Aufruf der Funktion einen

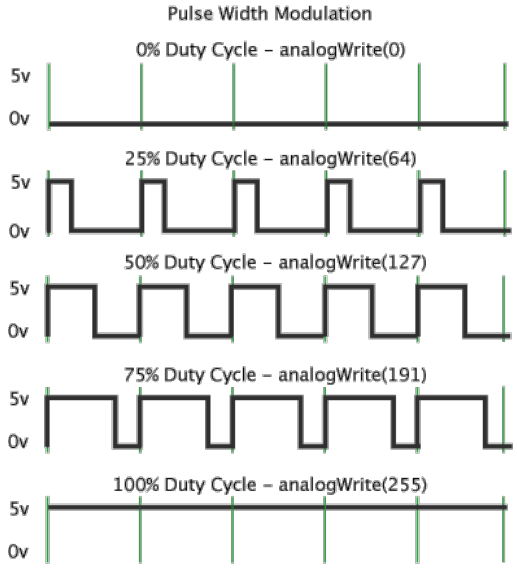


Abbildung 7: Die Pulsweitenmodulation von der Arduino Webseite. Hier wird verdeutlicht, wie die Dauer des Signals für An (5V) über die Zeit bestimmt wird. Quelle: [Hir13]

Wert inkrementiert, bis der Wert 100 erreicht ist. Daraufhin wird der Zähler auf 0 zurückgesetzt. Dieser Zähler dient als „Zeitscheibe“. Als Parameter übergeben wir eine Zahl, die als Prozentwert des Helligkeitswertes gewertet wird. Das wäre bei 100 ständig an, also 100% und bei 50 bis zur Hälfte der Zeit an und 0 für ständig aus. Mit einer Abfrage wird geprüft, ob der Zähler

(„Zeitscheibe“) größer ist, als der Wert der übergeben wurde. Wenn der Zähler kleiner als der übergebene Wert ist, dann werden alle LEDs angeschaltet, bei größer ausgeschaltet. Sehr gut verdeutlicht wird der Ablauf in der Abbildung 8.
Zur Absicherung, dass in dem Zeitraum die Sequenz nicht gestört wird, wird jedes Mal der Status der aktuellen LED-Sequenz abgefragt. So wird vermieden, dass zu einer dunklen Phase der Pulsweitenmodulation und hellen Phase der Sequenz, eine dunkle Phase am Ende entsteht und somit die Sequenz stört.

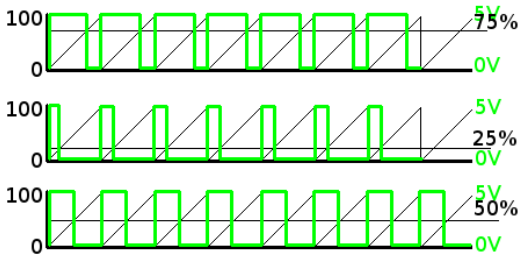


Abbildung 8: Der modellierte Pulsweitenmodulation, der auf allen Pins verfügbar ist. Links: Einheiten der Zeitscheibe von 0 bis 100. Grün: Der Zustand. schwarze Linie in der Mitte: Der übergeben Wert, gilt als Prozentwert.

3.2 Pin- und Sequenz-Verwaltung

Auf dem Arduino-Board werden Sequenzen in einem zweidimensionalen Array abgespeichert, der mit dem Wert 0 initialisiert wird. Damit der Arduino weiß, welche Sequenz für welche LED bestimmt ist, wurde ein Array für die LED-Pins eingerichtet. In dem LED-Array werden als Erstes die positiven Pins der LEDs eingetragen. Die entsprechende Pin-Nummer kann über die Arduino Due Webseite [Ard14c] herausgefunden werden. Danach gibt man an, wie viele Pins es insgesamt sind und die Sequenzlänge, hier 16-Bit. Aus den Werten wird dann ein Speicherbereich für die Sequenzen erstellt.
Beim Aufruf der Methode zum Setzen einer Sequenz übergibt man auch den Pin-Wert, der vorher eingetragen wurde. Daraufhin wird im LED-Array nach der Position des Pins gesucht. Wenn die Position gefunden wurde, wird diese Position im zweidimensionalen Speicher Array gesetzt und die Sequenz abgespeichert. Der Vorgang kann in der Abbildung 9 nachvollzogen werden.
In der Hauptfunktion, in der die Sequenzen an die richtigen LEDs übertragen werden, wird auf den zweidimensionalen Speicher zugegriffen. Da die Position im LED-Array und Sequenz-Speicher übereinstimmt, wird die richtige

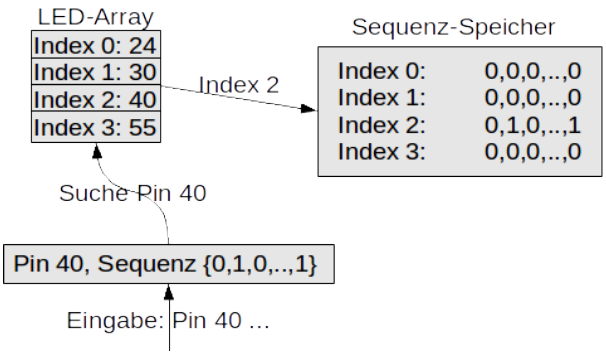


Abbildung 9: Unten links erfolgt die Eingabe für Pin 40. Danach wird der Index im LED-Array gesucht und auf dem Index im Sequenz-Speicher die Sequenz abgespeichert.

Sequenz und LED mit der Arduino Funktion „digitalWrite()“ angesteuert. Der Ablauf wird in Abbildung 10 gezeigt.

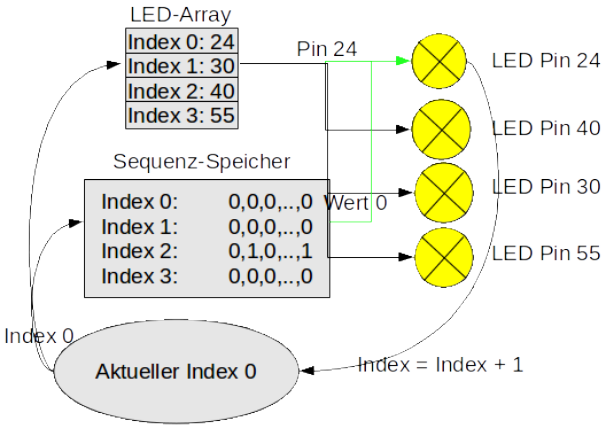


Abbildung 10: Der Speicher wird durchlaufen und die entsprechende LED wird angesteuert. Daraufhin wird der Index inkrementiert, bis alles LEDs durchlaufen wurden.

3.3 LED

Am Anfang der Entwicklung war die Überlegung zwei verschiedenfarbige LEDs zunehmen. So sollte eine Farbe für die Synchronisation dienen und eine weitere Farbe, die eigentliche Sequenz enthalten.
In der Abbildung 11 sieht man, dass es nicht wichtig ist welche Farbe man nimmt, die Kamera erfasst den Kern der LED als weiß, mit den BGR-Werten

(255,255,255). Da die Farbe wohl keinen großen Unterschied bei der Erkennung macht, wurde es dann mit diffusen und klaren LEDs versucht. Die klaren LEDs wurden recht schnell am Anfang aussortiert, denn das LED-Gehäuse reflektiert zu stark das eigene Licht, sodass zwei Punkte für eine LED zu sehen waren. Wir können in der Abbildung 12 klar die zwei Punkte für eine LED erkennen.

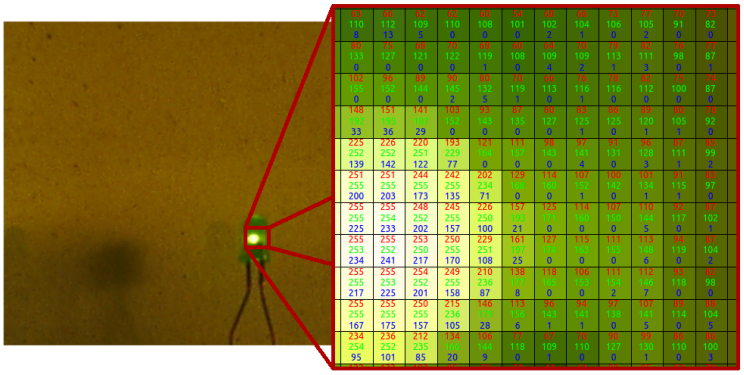


Abbildung 11: Grüne LED mit Anzeige der BGR-Werte

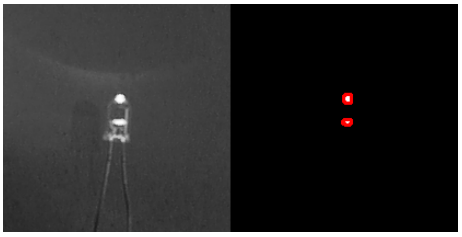
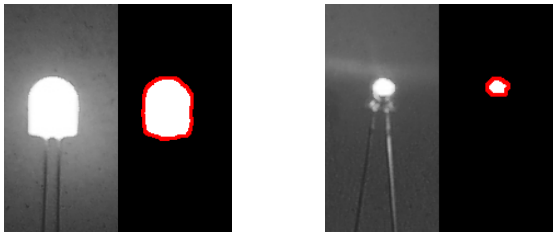


Abbildung 12: Klare LED. Links: Original. Rechts: Segmentierte Ansicht der Software.

Hingegen wird an einer diffusen LED das LED-Gehäuse gleichmäßig ausgeleuchtet und so ist die LED klar zu erkennen, wie man es in der Abbildung 13a sieht. Die diffuse LED hat den Vorteil, dass die LED von allen möglichen Richtungen gut erkannt wird, also einen sehr großen Abstrahlwinkel besitzt. Ein weiterer Vorteil ist, dass die LED größer wirkt, weil das komplette Gehäuse gleichmäßig ausgeleuchtet wird, hingegen bei der klaren LED nur der Kern zu sehen ist. So kann die diffuse LED bei einer größeren Entfernung noch gut erkannt werden. Am Ende wurde entschieden die LEDs mit einem klaren und kleinen Gehäuse zu nutzen, wie in Abbildung 13b zu sehen ist. Diese LED hat den Vorteil auch bei einer hellen Umgebung gut erkannt zu werden und besitzt einen Abstrahlwinkel von 170°. Der Durchmesser der

LED beträgt 4,8 mm, die Leuchtfarbe ist weiß und hat eine Lichtstärke von 800 mcd [Lt14].



(a) Diffuse LED (b) Kleine klare LED

Abbildung 13: Jeweils Links: Original. Rechts: Ansicht des Resultats der Segmentierung.

3.4 LED und Umgebung

Die LED-Helligkeit muss sich möglichst gut von der Umgebungshelligkeit abheben. Deshalb sind helle LEDs in einer hellen Umgebung und dunklere LEDs in einer dunkleren Umgebung am besten geeignet, da die hellen LEDs in einer dunklen Umgebung das Kamera-System blenden könnten.

3.5 Arduino Interrupt

Ein Arduino hat zwei vordefinierte Methoden, eine zum Initialisieren des Boards und eine Methode, in der der Programmcode so schnell wie möglich wiederholt wird. Da in der Wiederholungs-Methode der Helligkeitswert der dunklen Sequenz-Phasen geregelt wird, entstehen kleine Verzögerungen bei der Wiedergabe der Sequenzen. Da die Sequenzen am besten keine Verzögerung behindern sollte, wurde eine Timer-Methode aus dem Github Projekt von Ivan Seidel [Gom13] hinzugefügt, da die Bibliotheken aus Arduino-Playground [Ard14b] auf einem Arduino Due Board nicht ausgeführt werden konnten.
Der Timer Methode kann man eine Funktion übergeben und eine Frequenz, mit der die Methode ausgeführt werden soll. Diese Möglichkeit bietet eine zeitkritische Ausführung der Methode für die Sequenzdarstellung.

4 Sequenz

Um die LEDs untereinander zu unterscheiden, besitzt jede LED ihre einzigartige Folge von Impulsen. Diese Folgen oder Sequenzen, tragen keine direkten Informationen, erst das dazugehörige Computerprogramm erkennt die Bedeutung einer LED-Sequenz und kann die Sequenzen entsprechend zuordnen.

4.1 Aufbau

Die genutzte Sequenz ist eine Folge von Einsen und Nullen, die 16 Felder groß ist.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Wert	1	0	0	1	1	1	0	0	0	0	1	0	1	1	1	0

Es hat keinen genauen Grund, warum die Sequenz 16-Bit/Felder groß ist. Am Anfang war die Sequenz noch 8-Bit lang, diese Länge hatte den Vorteil, dass die Sequenz schnell erfasst wurde. Das Arduino-Board gibt die Sequenz mit 30 Hz wieder, die Kamera erfasst die Sequenz mit 60 Hz (doppelte Abtastrate) und so ergibt sich der Vorteil das in zwei Bilder die Phasen für dasselbe Bit stehen. Dieser Vorteil wird im nächsten Abschnitt in der Fehlererkennung genauer erklärt.
Die optimale Erkennungszeit zur Laufzeit beläuft sich je nach Länge der Sequenz. Das sind bei 8-Bit etwa $0,26s$ (1a) und bei 16-Bit etwa $0,53s$ (1b).

$$0,26\overline{6}s = 8 * \frac{1}{30}s \tag{1a}$$

$$0,53\overline{3}s = 16 * \frac{1}{30}s \tag{1b}$$

4.1.1 Hamming-Abstand

Der Nachteil einer kurzen Sequenz ist, dass nicht viele LED-Sequenzen erstellt werden können, da am besten ein möglichst großer Hamming-Abstand [Wik14a] genutzt werden soll. Der Hamming-Abstand gibt an, wie unterschiedlich zwei Bit-Sequenzen untereinander sind. In Tabelle 1 unterscheiden sich die zwei Sequenzen um 8 Werte, also ist der Hamming-Abstand 8. Deshalb war die Idee, den Hamming-Code dafür zu nutzen, der laut Wikipedia [Wik14b] einen Hamming-Abstand von mindestens 3 vorgibt. Dadurch

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Sequenz 1	0	0	1	0	0	1	1	0	0	1	0	1	1	1	1	0
Sequenz 1	1	0	0	1	1	1	0	0	0	0	1	0	1	1	1	0
Ungleich	1	0	1	1	1	0	1	0	0	1	1	1	0	0	0	0

Tabelle 1: Beispiele für ein Hamming-Abstand von 8 mit zwei Sequenzen.

können bei einem (7,4)-Hamming-Code $16 = 2^4$ Sequenzen generiert werden. Mit einem (15,7)-Hamming-Code lassen sich $128 = 2^7$ Sequenzen generieren. Da der Hamming-Code nur 15-Bit abdeckt und die Sequenz 16-Bit lang ist, wird der letzte Bit aus den generierten Prüfbits mit einer XOR Verknüpfung erstellt.
Die Sequenzen in der Tabelle 1, sind Sequenzen die mit einem Software-Tool von Dr. Norman Hendrich's generiert werden. Die Sequenzen werden dabei per Zufallswert generiert und auf Tauglichkeit überprüft, wie Hamming-Abstand und gleichmäßige Verteilung der 0 und 1 Werte.

4.2 Vorbeugung von dauerhaften Fehlern in der Sequenzerkennung

Die Kamera arbeitet im aktuellem Aufbau mit 60 Hz und das Arduino-Board gibt die Sequenz mit 30 Hz wieder, dieser Aufbau ermöglicht deshalb eine doppelte Abtastrate.
Durch die doppelte Abtastrate hat man die Möglichkeit Fehler bereits beim Empfangen zu erkennen. Die erste Fehlererkennung geschieht durch die Prüfung auf doppelte Bits. Dazu wird die 32-Bit lange Sequenz, bei einer 16-Bit Sequenz[3], in zwei 16-Bit Sequenzen gespaltet. Dazu wird die erfasste Sequenz Bit für Bit durchgegangen. Die Werte an ungerader Stelle kommen in das Odd-Array und die Werte an gerader Stelle kommen in das Even-Array. In der Tabelle 2 kann man sehen, dass in der Übertragung 2 Werte möglicherweise nicht korrekt erfasst wurden. Die falsch erfassten Werte werden korrigiert, indem als Erstes nach einer ähnlichen Sequenz gesucht wird -deshalb ist ein möglichst großer Hamming-Abstand hilfreich. Wenn die passende Sequenz gefunden wurde, werden die falsch erfassten Bits mit den Bits aus der originalen Sequenz überschrieben.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Erfasste Sequenz	1	1	0	0	1	1	1	1	0	1	0	0	0	1	0	0
Odd-Array	1	0	1	1	0	0	0	0								
Even-Array	1	0	1	1	1	0	1	0								
Fehler in der Sequenz	0	0	0	0	1	0	1	0								
Original Sequenz	1	0	1	1	1	0	0	0								
Erfasste Sequenz korrigiert	1	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0

Tabelle 2: Beispiel an einer 8-Bit Sequenz.

[3]Wegen begrenztem Platz, werden die Beispiele mit einem 8-Bit langer Sequenz erläutert. Die Beispiele kann man analog an 16-Bit lange Sequenzen oder anderen Längen anpassen.

5 Grafische Benutzeroberfläche

Das Programm nutzte am Anfang die Oberfläche aus der Diplomarbeit [Ric11], jetzt wird eine extra dafür überarbeitete grafische Benutzeroberfläche genutzt (siehe Abbildung 14). Der obere Teil des Hauptfensters stellt dabei die drei Kamerabilder dar, wobei die Darstellung zwischen den Originalbildern und den segmentierten Bildern, umgeschaltet werden kann. In der unteren Hälfte kann eine neue Sequenz für die Erfassung eingegeben werden und eine Aufnahme eines Versuchs erstellt werden.

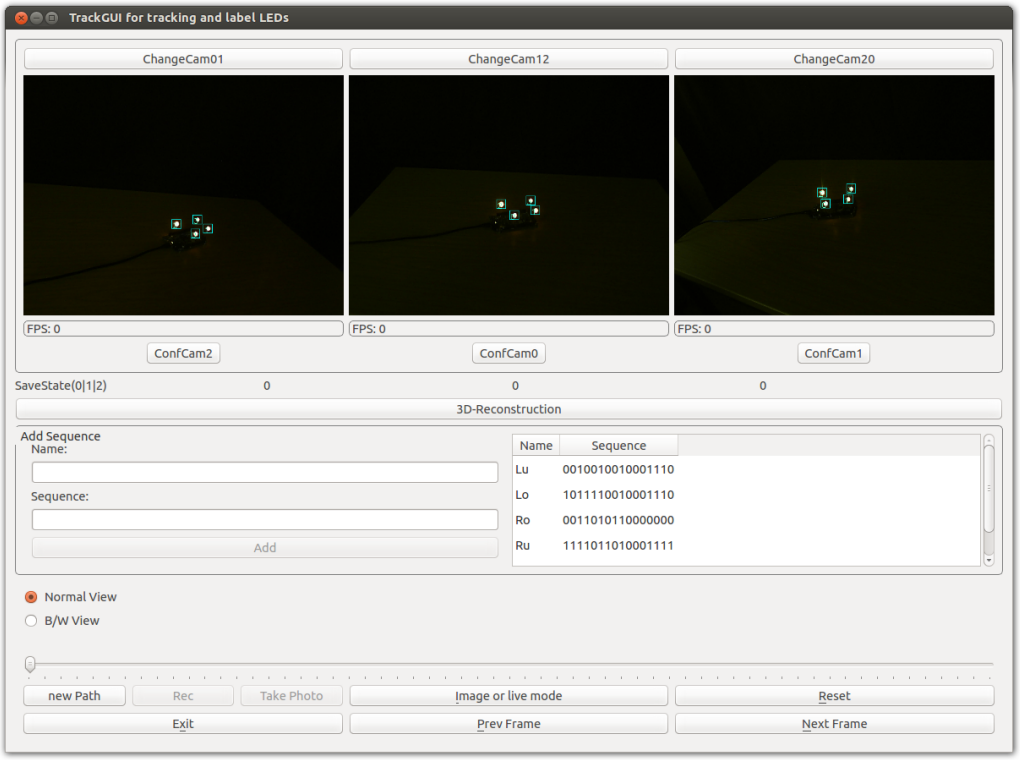


Abbildung 14: Hauptfenster. Im oberen Teil werden die Kamerabilder dargestellt und Positionen angepasst, in der Mitte die Sequenzen übergeben und im unteren Teil die Aufnahmen gemacht oder abgespielt.

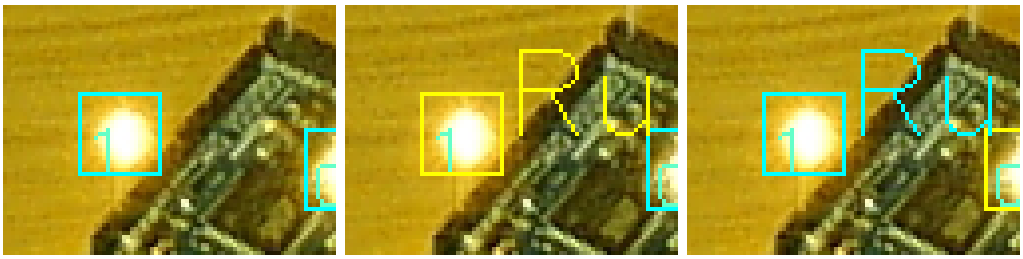
Da das Programm als eine Erweiterung der englischsprachigen Arbeit diente, sind die Knöpfe auf Englisch und die meisten Begriffe auch, aber eine Übersetzungsdatei liegt beim Programm bei. Die Anordnung der Kamerabilder können mithilfe den obigen Knöpfe zur Laufzeit getauscht werden. Hierfür steht „ChangeCam01“ zum Austauschen der Kamera 0 Position mit der Kamera 1 Position. Die anderen zwei Knöpfe folgen analog. Dann kommen die

jeweiligen Ansichten der Kamerabilder und direkt darunter kann die jeweilige Kamera konfiguriert werden, z.B. mit „ConfCam0“ für Kamera 0.
„SaveState(0|1|2)“ gibt an wie viele Bilder von der Aufnahme im Arbeitsspeicher sind und im Moment auf die Festplatte geschrieben werden. Nähere Informationen dazu sind auf der Seite 41 zu finden.
Der Knopf „3D-Reconstruction“ dient dazu, dem Programm mitzuteilen, dass die Kamerapositionen jetzt stimmen und die 3D-Rekonstruktion in einem OpenGL Fenster dargestellt werden kann.
Im Feld „Name:“ kann ein Name gewählt werden, dieser Name wird später neben den „Worker“ [4] und in der Tabelle rechts eingeblendet. Im nächsten Feld kann zu dem Namen, zugehörige neue 16-Bit lange Sequenz eingeben werden.
„Normal View“ dient zur Darstellung der originalen Kamerabilder mit „Workern“ darauf.
„B/W View“ dient zur Darstellung des vorverarbeiteten Bildes. Dies ist sehr hilfreich, wenn der Nutzer sehen möchte, worauf das Programm reagiert. Nähere Informationen hierzu werden auf der Seite 27 erklärt.
Die darauf folgenden Knöpfe sind allgemein für die Aufnahme- und Wiedergabefunktion da. Der Schieber ist für die Anpassung der Geschwindigkeit der automatischen Wiedergabe gedacht. Der Knopf „new Path“ erstellt ein neuen Dateipfad mit einem Ordnern für die Aufnahme. Nähere Informationen hierzu werden auf der Seite 41 erklärt. Der Knopf „Rec“ ist zum Starten und Beenden der Aufnahme. Der Knopf ist erst verfügbar, wenn ein Pfad erstellt wurde, dasselbe gilt auch für den Knopf „Take Photo“. Beim Knopf „Take Photo“ wird das aktuelle Kamerabild abgespeichert. „Image or live mode“ ist zum Umschalten zwischen den Kamerabildern und abspielen der abgespeicherten Aufnahmen. Im Bilder-Modus wird der „FPS“ Zähler zum Indexzähler der Fotos. „Reset“ ist zum Neustarten der „Worker“. Die gespeicherten Werte werden dabei gelöscht und neu initialisiert. „Exit“ dient zum Schließen der Programm Ansicht und beenden des Programms. „PrevFrame“ ist dafür da, um im Bilder-Modus das vorherige Bild darzustellen. „NextFrame“ ist dafür da, um im Bilder-Modus das nachfolgende Bild zu zeigen.

[4] Als **Worker** werden die Rechtecke im Bild bezeichnet, da sie im Programm die meiste Arbeit verrichten.

5.1 „Worker“

Ein „Worker“ ist im Bild ein rechteckiger Rahmen, der eine kleine Zahl am Rande hat, um sie zu unterscheiden. Die „Worker“ haben in der grafischen Oberfläche drei Zustände, die anzeigen in welchem Zustand sich ein „Worker“ befindet. Das ist einmal ein blauer Rahmen der aussagt dass beim „Worker“ noch keine Sequenz erkannt wurde. Dieser blaue Rahmen wird in der Abbildung 15a dargestellt. Der nächste Status wäre ein gelber Rahmen mit der Kennzeichnung der gefundenen Sequenz an der rechten Seite. Das wird in der Abbildung 15b gezeigt. Als letzter Status wäre ein blauer Rahmen mit einer Kennzeichnung, der für den Verlust einer Sequenz steht. Dieses wird in der Abbildung 15c veranschaulicht.



(a) Sequenz nicht erkannt. (b) Sequenz bekannt. (c) Sequenz verloren.

Abbildung 15: Die verschiedene Zustände der „Worker“.

5.2 Zusatz in der „B/W Ansicht“

In der schwarz/weiß Ansicht sieht man um den „Worker“ einen hellen Kreis, wenn die Sequenz bekannt ist, siehe Abbildung 16. Dies erscheint, um dem Nutzer zu zeigen, dass die „Worker“-Position an die Sequenz übergeben wurde und somit für andere Programmteile nutzbar ist.

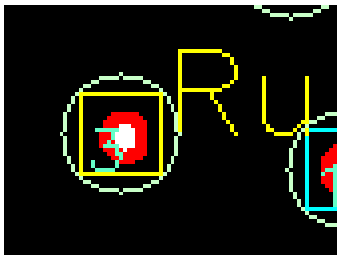


Abbildung 16: Darstellung eines „Worker“ mit einem hellen Kreis.

6 Kamerakalibrierung und die 3D-Rekonstruktion

Eine Kalibrierung der einzelnen Kameras ist notwendig, weil in der Herstellung von Hardware Abweichungen im Toleranzbereich nicht vermieden werden können. So kann es passieren, dass der Bildsensor der Kamera nicht genau mittig vor der Linse sitzt, sondern etwas versetzt ist und somit eine ungenau Darstellung der Szene liefert [Ope13a]. Die Linse kann eine Verzerrung des Bildes hervorrufen, die z.B. für den Zweck eines größeren Blickwinkels in Kauf genommen wird. Die fehlerbehaftete Darstellung der Szene ist für eine 3D-Rekonstruktion kontraproduktiv. Um die ungenaue Darstellung der Szene später im Verlauf zu korrigieren, werden die Kameras vorher kalibriert. Die dadurch erhaltenen Daten werden genutzt, um eine korrigiertes Abbild der Szene zu schaffen. Zusätzlich erhält man auch Informationen wie die Kameras zueinander ausgerichtet sind.
Die Kalibrierung für die 3D-Rekonstruktion wurde mithilfe der „Camera Calibration Toolbox for Matlab®“ [Bou13] bewerkstelligt. Mit diesem Matlab Tool wurde die notwendigen intrinsischen und extrinsischen Parameter bestimmt und in einer Datei festgehalten, damit sie später für das Programm genutzt werden kann. Die Erfassung und die Kalibrierung lief wie auf der Toolbox-Webseite [Bou13] beschrieben ab. Als Erstes wurden mehrere Stereo-Bilder von jeweils der linken & mittleren Kamera und dann von der mittleren & rechten Kamera aufgenommen, dabei wurde ein Schachbrettmuster, wie in der Abbildung 17 zusehen ist, vor der Kamera in verschiedenen Positionen gehalten. Die erfassten Bilder wurden dann mit den Matlab Tool eingelesen

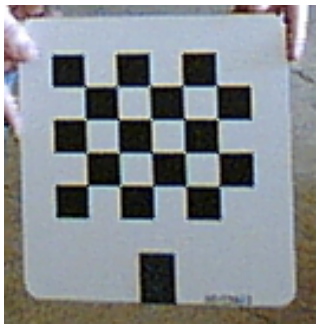


Abbildung 17: Schachbrettmuster 6x5

und ausgewertet. Daraus entstanden Daten wie etwa die Ausrichtung bzw. Positionierung des Bildsensores, wie die Krümmung der Linse das Bild beeinflusst und wie die Kameras zu der mittleren Kamera stehen, da die mittlere Kamera als Referenzpunkt für die 3D-Rekonstruktion dient (siehe Abbildung 18). Die erhaltenen Daten werden vom entwickelten Programm eingelesen,

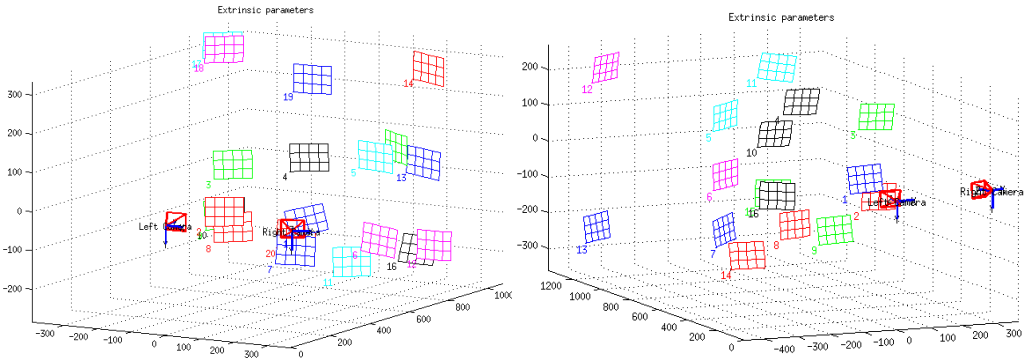


Abbildung 18: Übersicht der aufgenommenen Kalibrierungsmuster für beide Stereo-Paare des Kamerasystems.

an eine Funktion übergeben und die Kamerabilder entsprechend entzerrt. Im folgenden Kapitel werden die jeweilige Parameter genauer beschrieben. In der Abbildung 19 sieht man die drei Bilder des Kamerasystems, mit jeweils einer anderen Sicht zum Objekt. Daraufhin werden die Bilder entsprechend den aus der Kalibrierung bestimmten Parameter rektifiziert, wie in der Abbildung 20 zu sehen ist. Die Rektifizierung sorgt dafür das erfasste gleiche Objekte vor den Kameras auf der selben Zeilenposition bleiben, dies kann man gut an der Abbildung 21 sehen. Diese gedachte Linie heißt Epipolarlinie [Wik13a] und hilft der Ortsbestimmung des Objekts, wenn auf den ersten Bild das Objekt erfasst wurde, dann beschränkt sich die suche auf den zweiten Bild nur noch auf der Epipolarlinie. Somit wird die 3D-Rekonstruktion vereinfacht, indem nur noch auf einer Linie die Entfernung des Objekts bestimmt und daraufhin trianguliert wird.

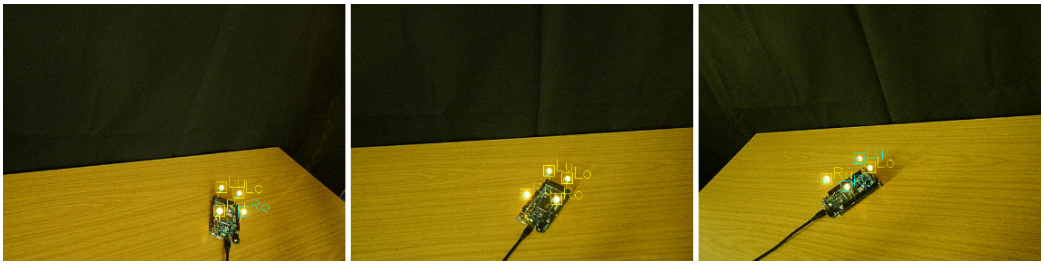


Abbildung 19: Normale Sicht der drei Kameras.

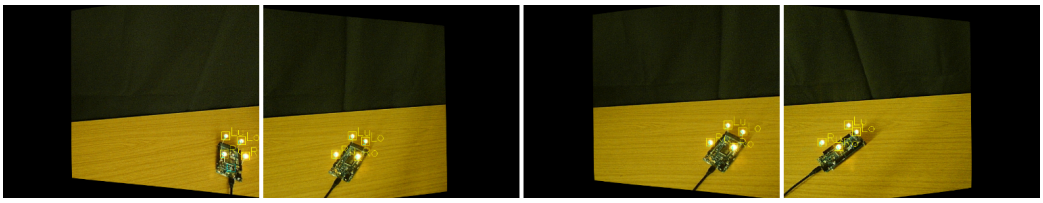


Abbildung 20: Rektifizierte Stereo-Paar Sicht (linkes Paar und rechtes Paar).

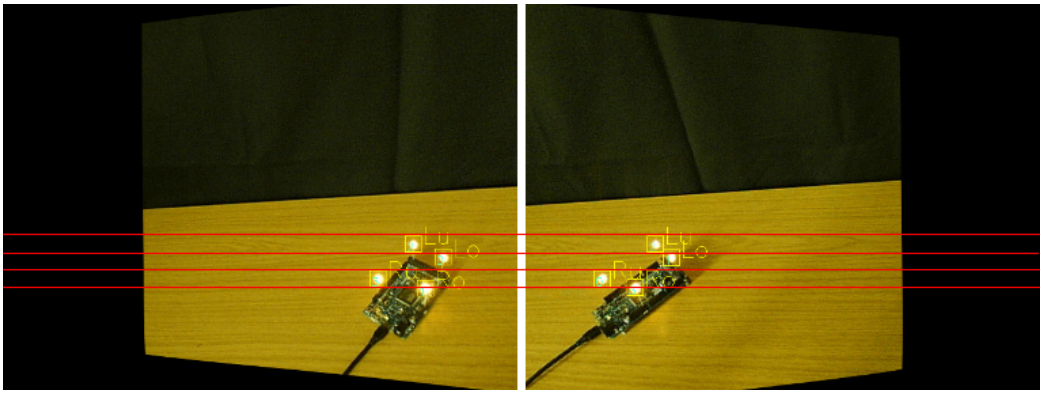


Abbildung 21: Ergebnis der Rektifizierung des rechten Stereo-Paars, mit Hilfslinien zur Darstellung der gleichen Zeilenposition der Objekte auf dem Bildern.

Die erfassten Informationen, welche LED an welcher Position ist, werden an den überarbeiteten Programm-Teil der Diplomarbeit [Ric11] übertragen, die daraus die dreidimensionalen Koordinaten errechnet. Da für die 3D-Rekonstruktion ein Stereo-Paar ausreicht und hierbei zwei Stereo-Paare betrieben werden, hat man einen größeren Erfassungsraum. Bei unverdeckter Sicht beider Paare auf die Marker besteht Redundanz, die genutzt werden kann um die Genauigkeit der rekonstruierten Koordinaten zu verbessern.
Die errechneten 3D-Koordinaten werden in eine Liste mit den Namen der Sequenz gespeichert. Die Liste wird dann an ein OpenGL-Fenster übertragen, das die Koordinaten in einem 3D-Raum darstellt, wie es in der Abbildung 22 zu sehen ist.

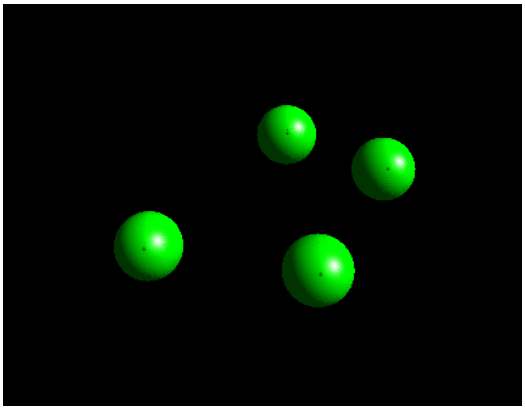


Abbildung 22: 3D-Rekonstruktionresultat nach der Erkennung und Zuordnung der LEDs.

Die Punkte werden in drei Grundfarben dargestellt. Grün steht für die erfolgreiche Erkennung mit geringer Abweichung der zwei Stereo-Paare. Rot steht für das linke Stereo-Paar und blau für das rechte Stereo-Paar. Man kann auch die erfolgreichen Punkte (grün) nach eigenen Vorstellungen färben. Dafür muss man nach den Namen der Sequenzen ein „,“ mit den jeweiligen RGB-Werten von 0 bis 1 und mit einem Komma getrennt übergeben. So steht „Name,0.5,0.5,0.5“ für die Farbe grau.

6.1 Ergebnisse der Kamerakalibrierung

Die erhaltenen Daten aus der Kalibrierung werden in zwei Dateien kopiert, jeweils für ein Kamerapaar. Dabei enthalten sind jeweils extrinsische und je Kamera intrinsische Kameraparameter [Hof09].
Die extrinsischen Parameter geben die Position und Orientierung der beiden Kameras eines Stereo-Paars zueinander an.

Parameter	X-Achse	Y-Achse	Z-Achse
Rotationsvektor (rad)	0,03061	0,50318	0,00774
Translationsvektor (mm)	-321,75168	-3,19734	79,65450

Tabelle 3: Die extrinsischen Kameraparameter für das linke Kamerapaar

Parameter	X-Achse	Y-Achse	Z-Achse
Rotationsvektor (rad)	0,2655	0,50203	-0,00165
Translationsvektor (mm)	-321,55	-2,15522	83,07042

Tabelle 4: Die extrinsische Kameraparameter für das rechte Kamerapaar

Für den verwendeten Aufbau werden die bestimmten extrinsischen Parameter in der folgenden Tabelle 3 (fürs linke Kamerapaar) und in der Tabelle 4 (fürs rechte Kamerapaar) gezeigt. Die intrinsischen Kameraparameter geben an, wie die 3D-Weltkoordinaten in 2D-Bildkoordinaten transformiert werden. In der Abbildung 23 wird die Transformation skizziert.
Für die Berechnung ist also wichtig zu wissen, wo der Bildmittelpunkt genau sitzt und wie weit die Brennweite dazu ist. Zusätzlich werden Informationen zu der Linse benötigt, da die Linse den Bildpunkt entsprechend nachteilig für die genaue Bestimmung der Weltkoordinate verzerrt.
Aus der Kalibrierung erhält man zwei Matrizen für jeweils eine Kamera. Das sind einmal die Kamera-Matrix mit den Pixelwerten für Brennweite und Bildmittelpunkt. Als weitere Werte wird die Linsenverzerrung der jeweiligen Kamera als radiale- und tangentiale Verzerrung beschrieben. In der Tabelle 5 sind die intrinsischen Kameraparameter fürs linke Kamerapaar und in der Tabelle 6 sind die intrinsischen Kameraparameter fürs rechte Kamerapaar dargestellt.

Die Ergebnisse werden vom Programm angenommen und daraus resultieren dann die entzerrten Bilder, wie sie schon in den Abbildungen 21 gezeigt wurden.

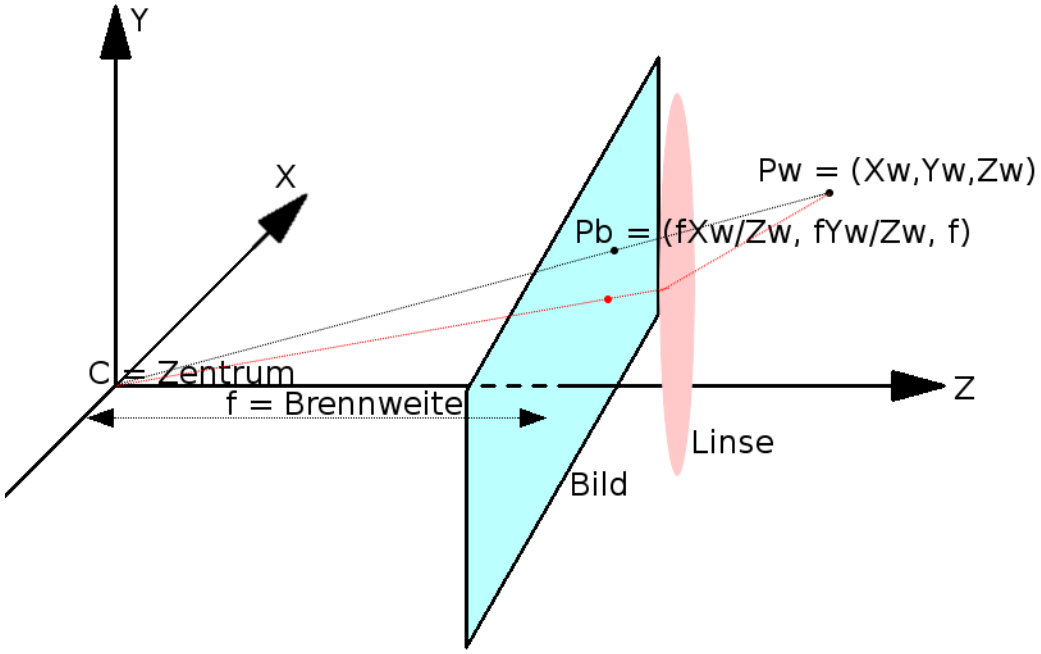


Abbildung 23: Darstellung der Transformation von Weltkoordinate Pw zur Bildkoordinate Pb. Die rote Linie zeigt dabei eine mögliche Linsenverzerrung an, die den eigentlichen Pb etwas versetzt.

Parameter	Linke-Kamera	Mittlere-Kamera
Brennweite (px)	[542,48379 ; 542,25396]	[544,43411 ; 544,46241]
Bildmittelpunkt (px)	[292,65992 ; 258,20888]	[317,13597 ; 236,18756]
Radiale Verzerrung	[-0,10549 ; 0,14675]	[-0,10759 ; 0,13459]
Tangentiale Verzerrung	[-0,00092 ; -0,00045]	[-0,00237 ; -0,00019]

Tabelle 5: Die intrinsischen Kameraparameter für das linke Kamerapaar

Parameter	Mittlere-Kamera	Rechte-Kamera
Brennweite (px)	[542,80471 ; 543,10652]	[543,7466 ; 543,74660]
Bildmittelpunkt (px)	[316,42889 ; 235,82381]	[331,74693 ; 228,42710]
Radiale Verzerrung	[-0,10879 ; 0,13786]	[-0,11299 ; 0,15634]
Tangentiale Verzerrung	[-0,00285 ; -0,00066]	[0,00078 ; -0,00139]

Tabelle 6: Die intrinsischen Kameraparameter für das rechte Kamerapaar

7 Echtzeit-Objekt-Verfolgung mit aktiven Markern

Die Verarbeitung wurde recht einfach gehalten, wie wir es in der Abbildung 24 sehen können.

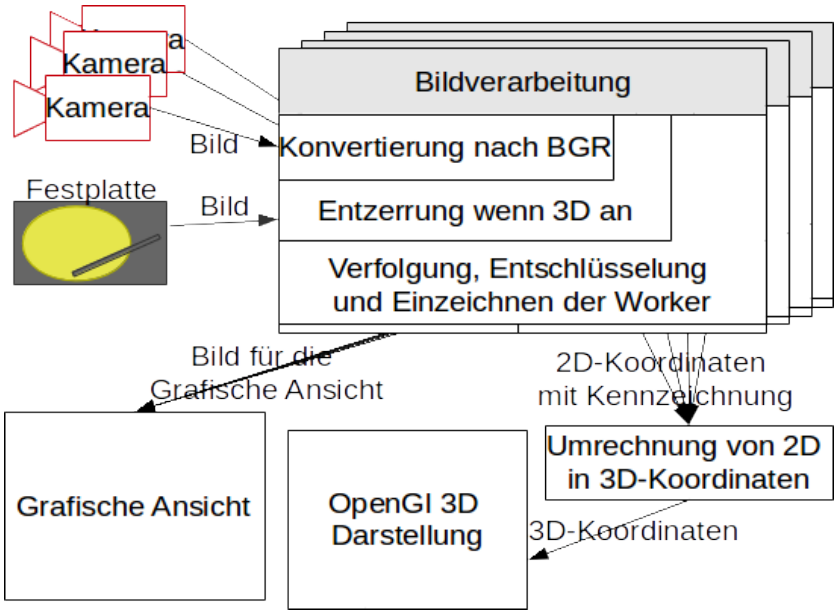
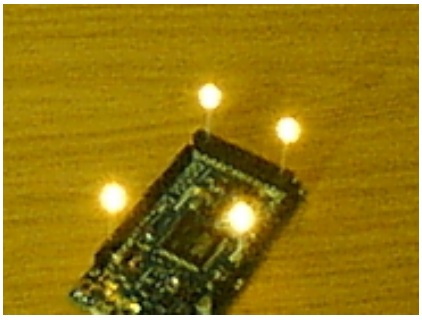


Abbildung 24: Verarbeitungsschritte vereinfacht dargestellt.

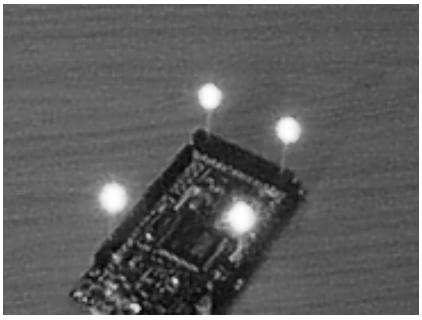
Je nach Einstellung kann das Kamerabild oder ein abgespeicherter Versuch von der Festplatte abgespielt werden. Das Kamerabild wird noch vorverarbeitet, wohingegen die Bilder von der Festplatte ab der BGR-Verarbeitung eingespielt werden können.
Da die PlayStation Eye Kamera die Bilder im YCbCr-Farbmodell [Wik14d] an den Computer überträgt, wird zunächst in das BGR-Farbmodell konvertiert. Je nach Einstellung wird dann das erfasste Bild für die 3D-Rekonstruktion entzerrt oder ohne Entzerrung an den Programmteil weitergeben, der für das Erkennen und Verfolgen zuständig ist. Die Entzerrung wird erst dann durchgeführt, sobald die Vorkonfiguration beendet wurde.
Die Vorkonfiguration beinhaltet das Austauschen der Kamera entsprechend ihrer Position in der Ansicht[5], dies kann beim Programmstart ausgeführt werden. Zusätzlich wird eine einmalige Berechnung der Kamerapositionen zueinander benötigt, die schon auf der Seite 21 beschrieben wurde.

[5]Vorher war es nötig die Reihenfolge der Kameras beim Anschließen an den Computer beizubehalten.

Als nächsten Schritt wird das Originalbild (siehe Abbildung 25a) in ein Graustufenbild konvertiert (siehe Abbildung 25b) und durchläuft daraufhin einen Funktion, der das Bild zu einem Binärbild konvertiert (siehe Abbildung 25c) dies geschieht mithilfe der OpenCV Bibliothek. Der Schwellwert für das Binärbild liegt dabei sehr hoch, da nur die leuchtenden LEDs erfasst werden müssen, siehe Abbildung 25c und die von Grund auf, auf dem Bildern als helle Flecken erscheinen, siehe dazu Abbildung 11.

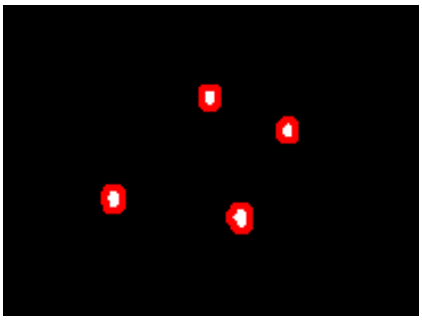


(a) Original (Zoom)

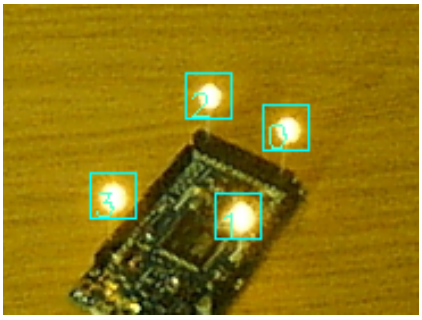


(b) Original ohne Farbe (Zoom)

(c) Binärbild (Zoom)



(d) Erkannte Objekte (Zoom)



(e) Original mit gesetzten „Workern“

Abbildung 25: Die einzelnen Verarbeitungsschritte.

Als nächster Schritt wird das Bild etwas geglättet, optisch zu sehen als eine verschwommene Sicht[6]. Durch die Glättung erreicht man das die Kanten weicher werden und die Konturensuche weniger Rauschen an den Kanten findet. Danach werden die LEDs mithilfe der OpenCV Funktion „findCountours“ gesucht, dabei bekommt man eine Liste von „Momente“[7] geliefert. Der Ursprungs-Code stammt von [Ope13b] ab und wurde entsprechend für diese Arbeit geändert.
Basierend auf den „Moments“ werden dann nur die Objekte in Betracht gezogen, die möglichst rund sind. Dafür wird dieselbe Formel (2) genutzt, die in der Diplomarbeit [Ric11] auf der Seite 28 beschrieben wird. Die Variablen für die Formel 2 stammen aus der OpenCV Webseite zu „Moments“ [Ope13d].

$$\varepsilon = \frac{mu_{20} + mu_{02} + \sqrt{(mu_{20} - mu_{02})^2 + 4 * mu_{11}^2}}{mu_{20} + mu_{02} - \sqrt{(mu_{20} - mu_{02})^2 + 4 * mu_{11}^2}} \tag{2}$$

Der Wertebereich der Exzentrizität ε liegt dabei in $[1, \infty)$, wenn der Wert nah an 1 liegt steht es für kreisrund, liegt der Wert außerhalb, ist das Objekt mehr oval [Wik13b].

Die von der Konturensuche gelieferte Liste von Momente werden dann für die Schwerpunkterrechnung genutzt. Die Schwerpunkterrechnung geschieht durch die Formel 3, die in der OpenCV Dokumentation zu den Momente [Ope13d] beschrieben wird.

$$x = \frac{m_{10}}{m_{00}}, y = \frac{m_{01}}{m_{00}} \tag{3}$$

Die akzeptierten Schwerpunkte werden in eine Liste gespeichert, daraufhin wird die Liste dann an eine Funktion übergeben, die zu jedem gefundenen Punkt einen passenden „Worker“ sucht.
Die Funktion geht als Erstes durch alle übergebenen Punkte und prüft, welcher der vorhergehenden „Worker“ zu den neuen Koordinaten besser passt. Dabei wird die vorletzte, letzte und mögliche zukünftige Position beachtet. In der Abbildung 26 wird dies durch die Rechtecke veranschaulicht. Die Suche der Positionen ist dabei nicht auf die Rahmen beschränkt, sondern eher auf die angedeutete weiße Fläche in der Abbildung 26. Da es keine harten Grenzwerte gibt, wird einfach der „Worker“ gesucht, der möglichst dicht an

[6] Im englischen auch als „blur“ bezeichnet, so heißt auch die OpenCV Funktion.
[7] **Momente** werden in der Bildverarbeitung genutzt um bestimmte Objekte auf einem Bild zu beschreiben [Wik13c].

Abbildung 26: Ein Ausschnitt nach der Konturensuche mit optischen Hilfsmitteln zur Erklärung und skizzenhaften Darstellung. Innerhalb der weißen Striche befinden sich die LEDs. Die „Worker“ werden ensprechend ihrer aktuellen Position und der LED im weißem Feld gesetzt.

den Schwerpunkten aus der Liste ist. Als Ausgangspunkt werden die Schwerpunkte aus der Liste herangezogen. Mit den Schwerpunkten aus der Liste werden die Entfernungen zu den einzelnen „Workern“ gemessen. Die Entfernung wird mit der Manhattan-Distanz (Formel 4) errechnet [Wik14c].

$$d(a, b) = |a_x - b_x| + |a_y - b_y| \tag{4}$$

Die vorletzte Position dient als Überprüfung, ob der „Worker“ die Position zu voreilig gewechselt hat, um dies zur Not im nächsten Schritt wieder korrigieren zu können. Das wird mit dem hellblauen „Letzte Pos.“ Rahmen dargestellt.
Die letzte Position dient zur Erkennung welcher „Worker“ schon in der Nähe des aktuellen Punktes ist und dient als Fortsetzung der Verfolgung. Dies wird mit den türkis-farbigen „Worker“ Rahmen dargestellt.
Die mögliche zukünftige Position wird durch die Vektorbildung (Formel 5) von der „vorletzten und letzten Position“ erreicht. Dabei wird die $np = neue\ Position$ errechnet, durch addieren des Stützvektors $ap = alte\ Position$ und des Richtungsvektors, welcher zuvor durch subtrahieren der ap und $vp = vorletzte\ Position$ berechnet wurde.

$$np(ap, vp) = ap + (ap - vp) \tag{5}$$

Die mögliche zukünftige Position wird in der Abbildung 26 durch den dunkelblauen Rahmen dargestellt.
Dadurch kann man festellen wohin die LED sich bewegen wird und ob ein „Worker“ schon in die Nähe der Erfassung kommt.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Erfasster-Wert	78	71	77	73	75	85	93	93	92	89	69	81	93	86	75	80
32-Bit Mittelwert	80	79	80	80	80	80	80	80	80	80	80	80	80	80	80	80
Temp-Array	0	0	0	0	0	1	1	1	1	1	0	1	1	1	0	0

Tabelle 7: Sequenz aus der Grafik bis zu 16 Bits entschlüsselt.

7.1 Erkennung

Da jedem „Worker“ seine jeweilige Position mittlerweile übergeben wurde, wird als Nächstes die Sequenz erkannt. Dabei wird für jeden „Worker“ eine kleine Fläche mit der entsprechenden LED vom ganzem Bild zugeteilt, gezeigt in der Abbildung 26. In dem zugeteilten 18x18 Pixel Feld wird jetzt ein durchschnittlicher Helligkeitswert berechnet und in einem 32-Bit großem Array abgespeichert. Mit den 32 vorherigen Helligkeitswerten kann man eine Schranke errechnen und mit dieser Schranke kann man dann sehen ob der neue aktueller Helligkeitswert eine dunkle oder eine helle Phase der Sequenz darstellt, siehe Abbildung 27 und Tabelle 7. Diese dunklen und hellen Phasen werden als 0 und 1 Werte in einen Zwischenspeicherarray von 32-Bit abgespeichert, und nur die ältesten Werte werden mit neuen Werten überschrieben.

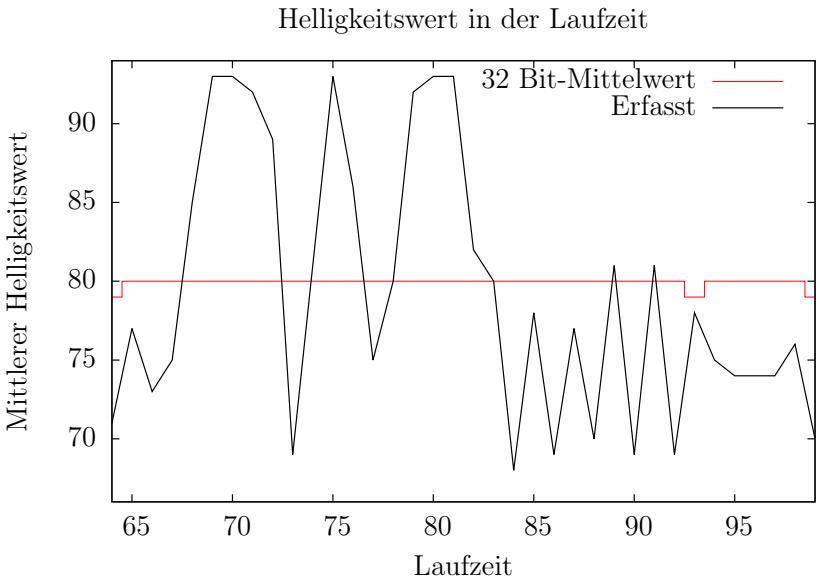


Abbildung 27: Sequenz Erfassung

7.2 Entschlüsselung

Die Entschlüsselung erfolgt, indem die Werte aus dem Zwischenspeicherarray auf jeweils 16-Bit lange Arrays aufgespalten werden. Da die Kamera mit einer Frequenz von 60 Hz arbeitet und das Arduino-Board die Sequenz mit 30 Hz anzeigt, erfasst die Kamera die Sequenz meistens doppelt. Bei der Aufspaltung in einen Odd- und Even-Array wird am Anfang noch geprüft, ob die Arrays übereinstimmen, das dient dazu mögliche Fehler zu beseitigen. Da die Sequenzen doppelt erscheinen, kann man schon erkennen wo ein Fehler auftrat.
Nach dem Aufspalten werden die erfassten Sequenzen mit den abgespeicherten Sequenzen verglichen. Da der Startpunkt einer Sequenz nicht bekannt ist, wird die Sequenz nach jeder Überprüfung um einen Wert verschoben gesucht. Zur Verdeutlichung sieht man in der Tabelle 8 eine Sequenz die um 2 Stellen verschoben ist.

Position	1	2	3	4	5	6	7	8
Gesuchte Sequenz	1	0	1	1	0	1	0	0
Seq. Hälfte Verschiebung (+0)	1	1	0	1	0	0	1	0
Seq. Hälfte Verschiebung (+1)	0	1	1	0	1	0	0	1
Seq. Hälfte Verschiebung (+2)	1	0	1	1	0	1	0	0

Tabelle 8: Beispiel an einer 8-Bit Sequenz. Eine Hälfte wird dargestellt, da die zweite identisch ist.

Wenn eine gesuchte Sequenz gefunden wurde, wird der Verschiebungsgrad zentral festgehalten, und die Position des „Workers“ wird an die gefundene Sequenz angeheftet. Durch den zentralen Verschiebungsgrad kann jeder neue und jeder aktuelle „Worker“ sich daran orientieren, somit hat die Software sich mit dem Arduino-Board synchronisiert.
Da die Sequenz jetzt gefunden wurde, wird die Position an die Sequenz geheftet, dass erleichtert später die Verarbeitung, da einfach nur eine Liste mit dem Sequenz-Namen und deren Position übergeben werden muss.
Wenn die erfasste Sequenz 1-Bit bis zu 2-Bit Fehler hat (im aktuellen Programm), wird als Erstes die Sequenz gesucht, die die größte Übereinstimmung mit einer gespeicherten Sequenz hat. Wenn die Sequenz mit der größten Ähnlichkeit gefunden wurde, dann werden die fehlerhaften Bits mit denen aus der Original-Sequenz überschrieben. Dies ist machbar, weil wir mit dem großen Hamming-Abstand ausschließen können, dass eine entsprechende andere Sequenz dazu passen könnte. Der Hamming-Abstand sollte auch möglichst in der Verschiebung groß bleiben, damit diese Möglichkeit der Erkennung funktioniert.

Bei erfolgreicher Erkennung einer Sequenz, wird beim „Worker“ ein Zähler für die entsprechende Sequenz hochgezählt. Das Hochzählen dient für zwei Dinge, erstens wenn die Sequenz nicht erkannt wurde, wird sie dennoch weiterhin verfolgt, und zweitens wenn eine andere Sequenz zufällig zu dem „Worker“ passt, wird dennoch die dominante Sequenz mit dem höheren Wert verfolgt. Wenn die Erkennung eine andere Sequenz findet, kann es durch das Rauschen der Kameraerfassung passiert sein, oder das Objekt wurde so schnell bewegt, dass die Kamera es nicht mehr erfassen konnte und dadurch nicht mehr dieselbe LED verfolgt.
Im Erfolgsfall kennen alle Sequenzen ihre jeweilige Position im Bild. Dieser Sequenz-Speicher wird dann an eine weitere Methode übergeben, die daraus die entsprechenden 3D-Koordinaten für jedes Objekt errechnet. Dies wurde schon im Kapitel 6 auf Seite 21 erklärt.

8 Ergebnisse und Performance

8.1 Geschwindigkeit bis zur ersten Erkennung

Die Kameras arbeiten mit 60 Hz, dabei kann man Bewegungen gut erfassen, solang die „Worker“ sich einmal synchronisiert haben. Dies geschieht am besten im Stillstand. Dafür müssen sich die Kameras als Erstes für etwa 0,550 Sekunden synchronisieren, etwa 33 Bilder mit einem Abstand von $\frac{1}{60}$ Sekunden.
In den ersten Sekunden wird der Grenzwert für die mittlere Helligkeit einer LED errechnet. In der Abbildung 28 sieht man dies durch die rote Linie[8] die ansteigt, bis sie bei 33 Bildern den Grenzwert erreicht hat. Ab dem Zeitpunkt, an dem der Grenzwert erreicht wurde, fängt die Sequenzerfassung an. Dazu werden die zuvor schon erfassten Werte einbezogen. Danach wird kontinuierlich die Sequenz weiter erfasst.
In der Abbildung 29 sieht man genau, dass es 33 Bilder gedauert hat, bis

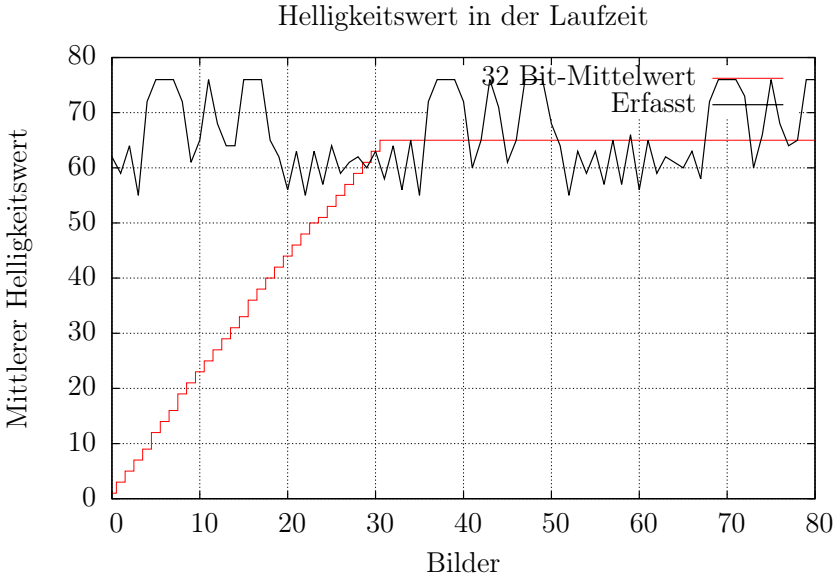


Abbildung 28: Erste Schritte der Sequenzerfassung.

die Verfolgung mit der entsprechenden Sequenz erfolgt.
Um die Sequenz nicht voreilig zu entschlüsseln, wurde für die ersten 32 Bilder in der ein „Worker“ erschaffen wurde, eine Sperre eingerichtet. Erst nach 32 Bildern kann der „Worker“ die Sequenz auswerten. Dies wurde gemacht, da in den ersten Millisekunden sehr häufig die falsche Sequenz erkannt wurde.

[8] Die Abbildung bezieht sich auf einem „Worker“, in diesem Fall auf die Nummer 0.

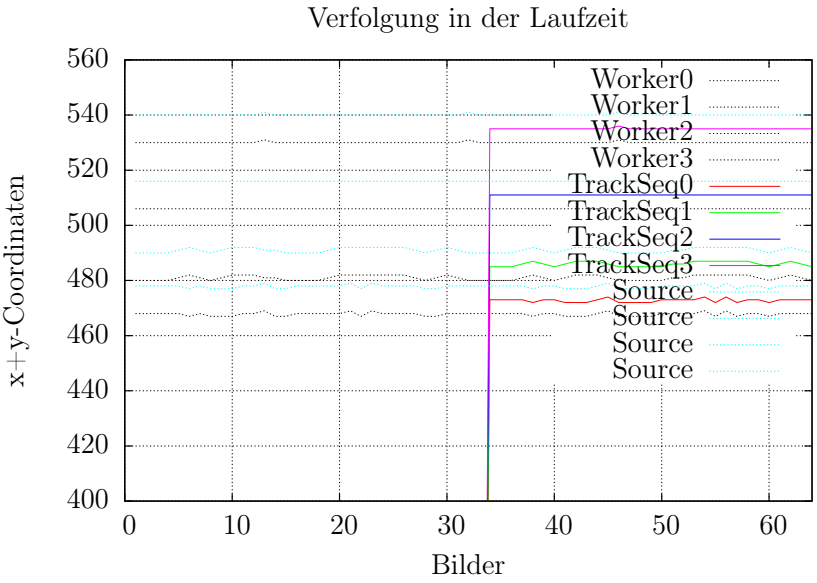


Abbildung 29: Sequenzverfolgung am Anfang. Wichtig zu erkennen das TrackSeq ab etwa 33 Bilder die Position sich ändert. Die X- und Y-Werte wurden addiert um eine Zeitliche 2D-Darstellung zu ermöglichen.

8.1.1 Steigerung der Geschwindigkeit

Die Erfassungszeit hängt vom Errechnen des Grenzwertes ab, die wiederum von der Sequenzlänge abhängt oder der Arbeitsgeschwindigkeit der Kamera. In den Beispielen war es ein 32-Bit großes Array, d.h. dass etwa bis zu dieser Anzahl die Arrays gefüllt werden müssen, bis ein guter Grenzwert errechnet wurde und ab da die 0 und 1 Phasen erkannt werden können.
Da die Methode nach jedem Bild ausgeführt wird, kann die Erkennung beschleunigt werden, d.h. wenn die Kamera mit 120 Hz läuft und der Arduino entsprechend schneller die 16-Bit lange Sequenz (in 60 Hz) darstellt, dann dauert es am Ende nicht mehr 0,550 Sekunden, sondern bei 33 Bildern etwa $0,275s = 33\ Bilder * \frac{1}{120}s$.

8.2 Verfolgung nach Verdeckung

Nach Verdeckung und Aufdeckung dauert es etwa 33 Bilder, bis eine erfolgreiche Verfolgung erneut stattfinden kann. In der Tabelle 9 sind die Werte aus einem Versuchsablauf dargestellt, die es verdeutlichen.

Zu sehen	Dauer in Bildanzahl	60 Hz
Verdeckt bis „Worker“ entfernt wird	nach 5-7 Bildern	116,66 ms
Aufgedeckt bis Sequenz erkannt wird	nach 33 Bildern	550 ms
Summe	38-40 Bildern	666,66 ms

Tabelle 9: Wiederaufnahme der Verfolgung nach Verdeckung.

8.3 Verfolgung und Stabilität

Das in dieser Arbeit entworfene Programm ist in der Lage die Marker stabil zu verfolgen, sofern die Bewegungen nicht zu ruckartig erfolgen und eher behutsam und ruhig ausgeführt werden. Da die Kamera im aktuellen Aufbau nur mit 60 Hz arbeitet, werden bei zu schnellen Bewegungen aus den runden LEDs längliche ovale Objekte, die die Software nicht mehr verfolgt. In der Abbildung 30 können wir sehen, dass die Verfolgung recht stabil abläuft. Um die X-Y-Koordinaten und Laufzeit zweidimensional darzustellen, wurden die X- und Y-Werte auf der Y-Achse addiert. Zusätzlich wurden für die verschiedenen Gruppen ein Offset von jeweils +5 und +10 in „TrackSeq“ und „Worker“ addiert, damit die Werte sich nicht gegenseitig auslöschen. So können wir sehen, dass die vier „Worker“ die meiste Zeit ihre entsprechende LED verfolgen.

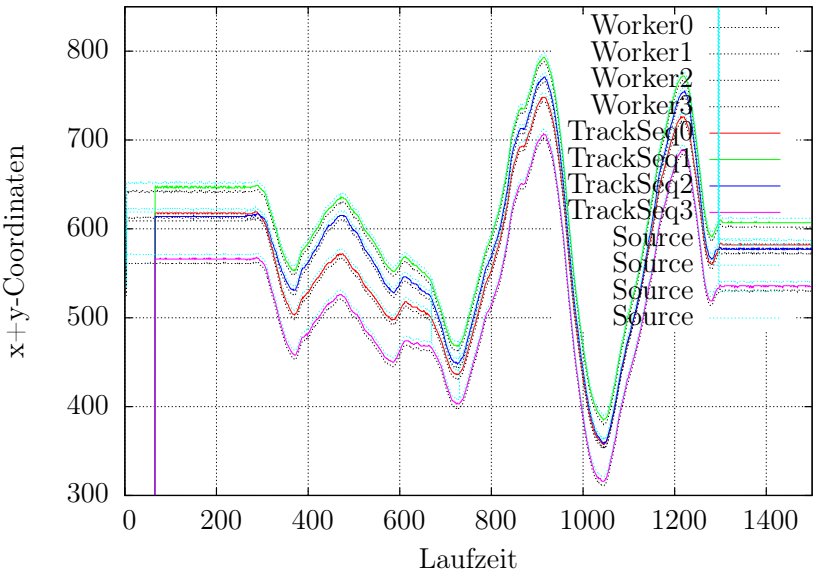


Abbildung 30: Eine komplette Sequenzverfolgung. Hierbei sieht man, dass die farbigen Linien ihre Ursprungswerte bis zum Ende verfolgen. D.h. dass jeder „Worker“ dessen entsprechende LED verfolgt hat.

8.3.1 Probleme der Helligkeitswerte

Das schnelle Vor- und Zurückbewegen des Objekts stört die Erkennung, da der Grenzwert sich immer an die Helligkeit anpassen muss und eine 60 Hz Kamera etwa 0,5 Sekunde braucht (siehe Seite 34) bis die Anpassung erfolgt. In Abbildung 31 ist gut sichtbar, dass der Grenzwert außerhalb der erkannten Sequenz liegt und somit für den „Worker“ den Anschein einer 0 Sequenz vorliegt.

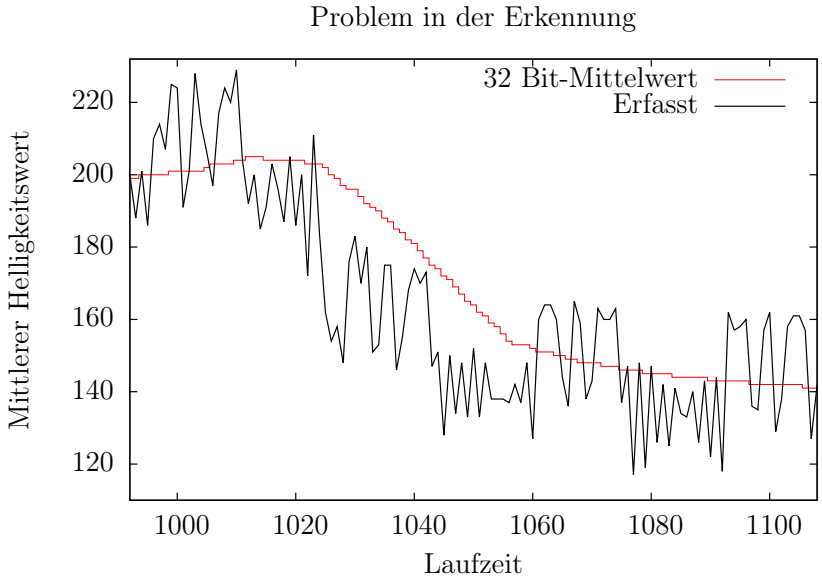


Abbildung 31: Helligkeitswert ausserhalb der Erfassung

8.4 Genauigkeit

Zum Testen der Genauigkeit der Positionsbestimung wurden die bekannten Abstände der LEDs auf dem Prototypobjekt mit den rekonstruierten Abständen verglichen.
Die Erkennung erfasst je nach Genauigkeit de kalibrierten Kameraparameter das Objekt genau. Für die Bestimmung der Genauigkeit wurden 1600 Bilder genutzt. Dabei befand sich das Objekt an verschieden Positionen. Aus den Werten, die nach der 3D-Rekonstruktion entstanden, wurden die entsprechenden Mittelwerte und Standardabweichungen errechnet. Zum Vergleich sind in der Abbildung 32 die Abmessungen der LEDs eingezeichnet und deren Kennzeichnung, wie etwa „Lo“ für „Links oben“.
So sehen wir in der Tabelle 10, dass das Programm präzise für den normalen Gebrauch ist, denn die Abweichungen beträgt nicht größer als $\pm 1\ mm$.

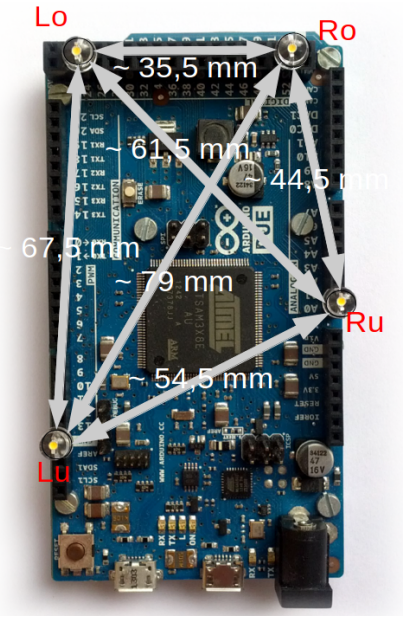


Abbildung 32: Arduino mit LED, Kennzeichnung und Abmessungen der LED Entfernungen zueinander.

Strecke	Mittelwert(mm)	Standardabweichung(mm)
Lo-Ro	35,7281	0,5705
Lo-Ru	61,6371	0,6213
Lu-Lo	67,7553	0,5821
Lu-Ro	78,9600	0,5582
Lu-Ru	54,3715	0,5233
Ru-Ro	44,2911	0,6523

Tabelle 10: Präzision der 3D-Rekonstruktion.

8.5 Performance

Die Performance des Programms hängt von vielen Faktoren ab, wie etwa der Prozessorleistung, der verfügbare Ressourcen, Bildfrequenz sowie weiterer Faktoren, die für die Computer Performance verantwortlich sind. Die hier getesteten Beispiele zeigen, dass das Programm mit einer 60 Hz Kamera Echtzeit-fähig ist. Da wir in der Summe für die Verarbeitung laut der Tabelle 11 unter 16,66 ms kommen und 60 Hz etwa 16,66 ms sind.

Ausführung	Mittelwert(ms)	Standardabweichung(ms)
Kamerabild aufnehmen	9,413	1,645
YCbCr zu RGB Konvertierung	2,694	0,5768
RGB zu BGR Konvertierung	0,726	0,1467
Identifizierung und Verfolgung	3,8179	0,6775
Summe	16,6512	1,8673
2D Punkte zu 3D Punkte	0,0209	0,0210

Tabelle 11: Zeit in ms nach etwa 1600 Bildern. Da die Umrechnung der 2D in 3D-Koordinaten auf einem anderen Prozessorkern läuft, wird der Wert nicht innerhalb der Tabelle erfasst.

9 Optionale Programmfunktionen

Während der Entwicklung und Ausarbeitung des Programms entstanden optionale Programmfunktionen, die hier beschrieben werden.

9.1 Aufnahme

Mit der Aufnahmefunktion können Experimente aufgenommen und später Schritt für Schritt wiedergegeben werden. Zum Abspeichern eines Experimentes wurde eine Funktion erstellt, die die Kamerabilder nach der Konvertierung abspeichert. Es werden also die Bilder abgespeichert, die ohne großen Aufwand gut zur Verarbeitung genutzt werden können und von Menschen visuell begutachtet werden können. Dazu wurde ein Stack-Speicher nach dem FIFO-Prinzip (First In - First Out) erstellt, dadurch werden neue Bilder am Ende hinzugefügt und ältere als Erstes abgespeichert. In der Abbildung 33 wird dieser Vorgang veranschaulicht. Diese Funktion arbeitet auf

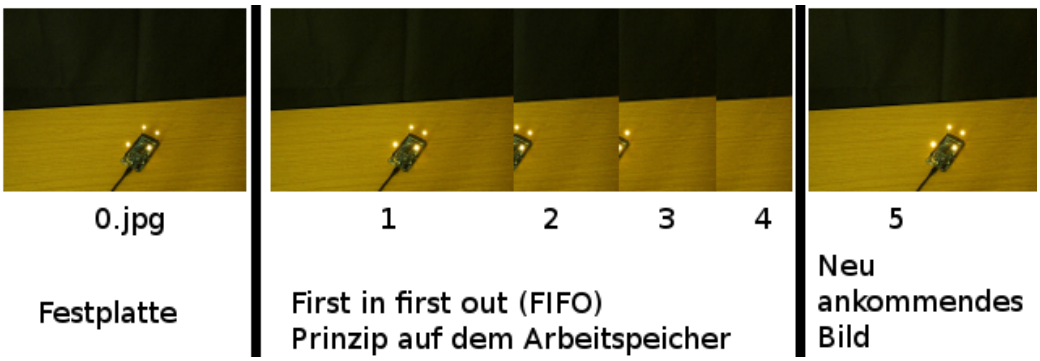


Abbildung 33: Skizzenhafte Darstellung des Speichervorgangs eines aufgenommenen Bildes.

einem anderen Prozessor, dadurch werden die anderen Verarbeitungen wie die Erkennung und Verfolgung ohne Verzögerung ausgeführt. Da der Zwischenspeicher temporär im Arbeitsspeicher erstellt und gehalten wird, wird der Computer beim Abspeichern nicht von der möglicherweise langsamen Festplatte ausgebremst. Dadurch entsteht der Vorteil, dass das Programm bei etwas leistungsschwächeren PCs, aber einen großen Arbeitsspeicher besitzen, mit dem Abspeichern der Bilder gut zu Recht kommt.

9.2 Wiedergabe

Die Wiedergabe erfolgt, indem die Bilder von einem vorhergehenden Experiment Schritt für Schritt eingelesen und wiedergegeben werden.

9.3 Datenstruktur

Die erstellten und genutzten Daten befinden sich auf derselben Ebene des Programms. Wenn eine Aufnahme vorhanden ist, dann existiert eine „info.xml“ Datei und ein Ordner mit einem Datum als Name. Dieses Datum gibt an, wann das Experiment aufgenommen oder erstellt wurde und befindet sich auch in der Datei „info.xml“.
Für den vollen Funktionsumfang braucht man die Kalibrierungsdaten, wie auf der Seite 21 beschrieben. Diese Daten müssen im YML-Format von OpenCV vorliegen und jeweils die Namen „cameracaliLM.yml“ fürs linke Kamerapaar und „cameracaliMR“ fürs rechte Kamerapaar tragen. Dann müssen sie noch, je nach Name, die „camera_matrixL“, „distortion_coefficientsL“ für linke Kamera und „camera_matrixM“, „distortion_coefficientsM“ für mittlere Kamera haben. Zusätzlich den „rotation_vector“ und „translation_vector“ der jeweiligen Paare zueinander besitzen.
Wenn Aufnahmen von älteren Projekten abgespielt werden sollen, sollte man die „info.xml“ entsprechend anpassen.

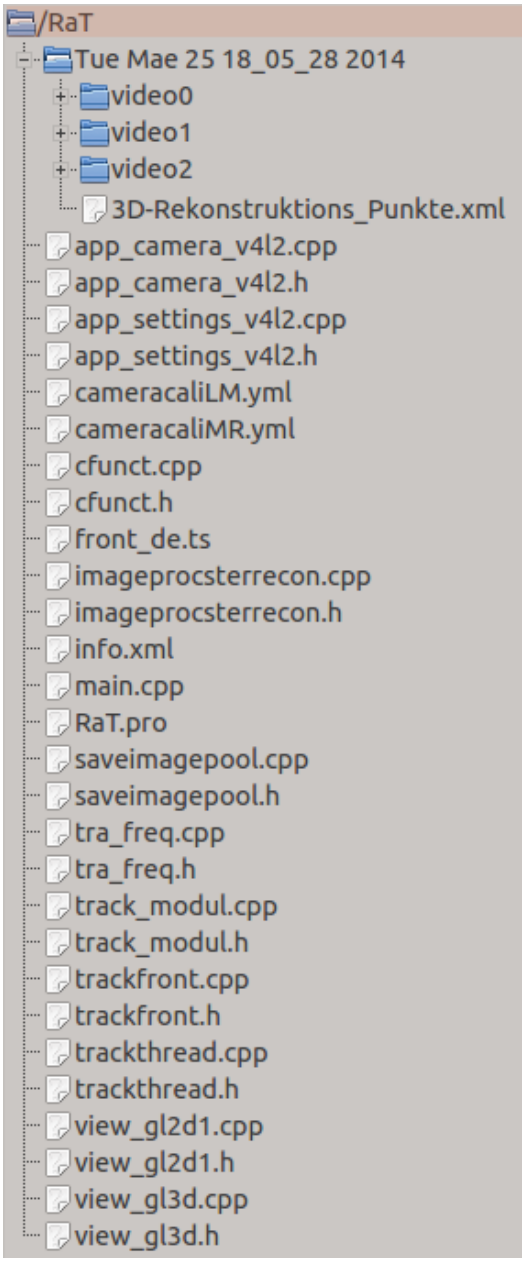


Abbildung 34: Ordnerstruktur

9.4 Infodatei (info.xml)

Die *info.xml* Datei wird beim Erstellen eines neuen Pfades aktualisiert oder erstellt. In der Datei befinden sich die Informationen zu den Speicherpfaden, die mit dem Programm erstellt wurden. Diese Informationen dienen dazu aufgenommene Versuche einfacher mit dem Programm zu verwalten und auszuführen.

9.5 Zeitstempel (timestamp.txt)

Jedes abgespeicherte Bild wird mit einer inkrementierten Zahl als Dateiname abgespeichert. Die Information, wann ein bestimmtes Bild gemacht wurde, wird in einer Textdatei namens *timestamp.txt* abgespeichert, die sich im selben Ordner wie die Bilder befindet. Dabei wird in der Datei eine fortlaufender Index und die zugehörige Zeit in ms Zeilenweise abgespeichert.

9.6 3DKoordinaten (3D-Rekonstruktions_Punkte.xml)

Die Punkte aus der 3D-Rekonstruktion werden im Wurzelverzeichnis des Aufnahmeordners gespeichert. In der XML-Datei werden für jedes aufgenommene Bild die 3D-Koordinaten gespeichert. Dabei wird als Erstes der Bildindex angegeben und dann die Anzahl von 3D-Punkten im aktuellem Bild. Daraufhin werden die Koordinaten mit den Namen der Sequenz und ihren X-,Y- und Z-Koordinaten abgespeichert.

10 Zusammenfassung

Diese Arbeit zeigt, dass es möglich ist mit handelsüblicher Hardware, Sequenzen von LEDs zu erfassen, sie zu verfolgen, in einem 3D-Raum darzustellen, und das alles in Echtzeit.
Dabei werden die Kamerabilder unabhängig voneinander verarbeitet, damit sie auf verschiedenen Prozessorkernen parallel ausgeführt werden können. Dies ermöglicht die Echtzeitverarbeitung. Zugleich ist das ganze System modular aufgebaut, sodass es theoretisch ohne großen Aufwand auf mehr als drei Kameras erweiterbar ist.
Die Sequenzen für die LEDs, die der Arduino Due modelliert, sind dabei 16-Bit lang und unterscheiden sich minimal um 6-Bits. Je größer die Unterscheidung, desto leichter fällt es dem System die Sequenzen zu unterscheiden. Zurzeit ist das Einsatzgebiet des Systems auf Innenräume ohne direkte Sonneneinstrahlung ausgelegt.

10.1 Zukünftige Arbeiten

In dieser Arbeit wurde viel in einer kurzen Zeit erreicht, aber es bleiben viele Möglichkeiten offen. Diese werden hier kurz erfasst und angesprochen.

- Kalibrierung über die Software vereinfachen oder automatisieren, da es zur Zeit recht aufwendig ist jede einzelne Kamera zu kalibrieren.
- Benutzerfreundliche Gestaltung der Software, wie etwa beim Abspeichern eines Versuches auch die LED Sequenzen entsprechend abzuspeichern und die Kamerapositionen hinterlegen.
- Die Möglichkeit mehr als 3 Kameras für die 3D-Rekonstruktion zu nutzen.
- Die Möglichkeit den Arduino direkt zu beeinflussen, wie etwa die Helligkeit je nach Umgebung automatisiert anzupassen.
- Ein Handschuh für das System zu entwickeln, um einer Handerfassung und Rekonstruktion zu bieten.
- Optimierung der Sequenz um in der Verdeckung von einer anderen LED, die verdeckte LED herauszufiltern.
- Automatische Generierung von Sequenzen, je nach Gegebenheit.
- Möglichkeit mehrere Arduino-Boards an einem System zu nutzen.

Literatur

[Ard14a] Arduino: *Arduino Board Due*. http://arduino.cc/en/Main/ArduinoBoardDue. Version: 2014

[Ard14b] Arduino: *Hallo Welt! - Mit Interrupt und Timer Library*. http://playground.arduino.cc/Deutsch/HalloWeltMitInterruptUndTimerlibrary. Version: 2014

[Ard14c] Arduino: *SAM3X-Arduino Pin Mapping*. http://arduino.cc/en/Hacking/PinMappingSAM3X. Version: 2014

[Arg13] Argyros, Antonis: *Efficient model-based 3D tracking of hand articulations using Kinect*. http://users.ics.forth.gr/~argyros/research/kinecthandtracking.htm. Version: 2013

[Bou13] Bouguet, Jean-Yves: *Camera Calibration Toolbox for Matlab*. http://www.vision.caltech.edu/bouguetj/calib_doc/. Version: December 2013

[Gom13] Gomes, Ivan S.: *DueTimer*. https://github.com/ivanseidel/DueTimer. Version: March 2013

[Hir13] Hirzel, Timothy: *PWM*. http://arduino.cc/en/Tutorial/PWM. Version: 2013

[Hof09] Hofmann, Tim: *Geometrische Kamerakalibrierung*. http://www.mi.hs-rm.de/~schwan/Projects/CG/CarreraCV/doku/intrinsisch/intrinsisch.htm. Version: 2009

[Lt14] LED-tech: *LED-tech Shop*. http://www.led-tech.de/de/Leuchtdioden/4.8mm-LEDs/Wide-Beam-LED-LT-1046_1_89.html. Version: 2014

[OKA11] Oikonomidis, Iason ; Kyriazis, Nikolaos ; Argyros, Antonis A.: Efficient model-based 3D tracking of hand articulations using Kinect. In: *BMVC*, 2011, S. 1–11

[Ope13a] OpenCV: *Camera calibration With OpenCV*. http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html. Version: 2013

[Ope13b] OpenCV: *Image Moments*. http://docs.opencv.org/doc/tutorials/imgproc/shapedescriptors/moments/moments.html#moments. Version: 2013

[Ope13c] OpenCV: *OpenCV (Open Source Computer Vision Library)*. http://opencv.org/. Version: 2013

[Ope13d] OpenCV: *Structural Analysis and Shape Descriptors*. http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=moments#moments. Version: December 2013

[Pha12a] PhaseSpace: *IMPULSE X2*. http://www.phasespace.com/impulse_motion_capture.html. Version: 2012

[Pha12b] PhaseSpace: *IMPULSE X2 motion capture system*. http://www.est-kl.com/fileadmin/media/pdf/PhaseSpace/PhaseSpaceBrochure2012.pdf. Version: 2012

[Pro13] Project, Qt: *Qt Project*. http://qt-project.org/. Version: 2013

[Ric11] Richter, Eugen: *Hand Pose Reconstruction using a Three-Camera Stereo Vision System*, University of Hamburg, diploma thesis, August 2011

[Son13] Sony: *PlayStation Eye*. http://de.playstation.com/ps3/peripherals/detail/item78898/PlayStation%C2%AEEye/. Version: 2013

[Wik13a] Wikipedia: *Epipolargeometrie*. http://de.wikipedia.org/wiki/Epipolargeometrie. Version: 2013

[Wik13b] Wikipedia: *Exzentrizität (Mathematik)*. http://de.wikipedia.org/wiki/Exzentrizit%C3%A4t_(Mathematik). Version: 2013

[Wik13c] Wikipedia: *Moment (Bildverarbeitung)*. http://de.wikipedia.org/wiki/Moment_(Bildverarbeitung). Version: 2013

[Wik13d] Wikipedia: *Passive markers*. http://en.wikipedia.org/wiki/Motion_capture#Passive_markers. Version: 2013

[Wik13e] Wikipedia: *Time modulated active marker*. http://en.wikipedia.org/wiki/Motion_capture#Time_modulated_active_marker. Version: 2013

[Wik14a] Wikipedia: *Hamming-Abstand*. http://de.wikipedia.org/wiki/Hamming-Abstand. Version: 2014

[Wik14b] Wikipedia: *Hamming-Code*. http://de.wikipedia.org/wiki/Hamming-Code. Version: 2014

[Wik14c] Wikipedia: *Manhattan-Metrik*. http://de.wikipedia.org/wiki/Manhattan-Metrik. Version: 2014

[Wik14d] Wikipedia: *YCbCr-Farbmodell*. http://de.wikipedia.org/wiki/YCbCr-Farbmodell. Version: 2014

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen benutzt habe, die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht

Datum, Unterschrift