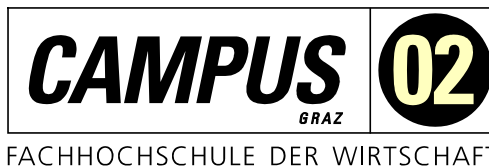


**Masterarbeit**

# **OPTISCHES AUSWERTESYSTEM FÜR STEELDARTSCHEIBEN**

ausgeführt am



Fachhochschul-Masterstudiengang  
Automatisierungstechnik-Wirtschaft

von

**Dominikus Weleba BSc.**

2010322023

betreut und begutachtet von  
FH-Prof. Dipl.-Ing. Dieter Lutzmayr

Graz, im Mai 2022

.....  
Unterschrift

## **EHRENWÖRTLICHE ERKLÄRUNG**

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....  
Unterschrift

## **DANKSAGUNG**

Ich möchte mich bei meiner Verlobten, Nanna Schickhofer, bedanken, die mich nicht nur bei dieser Arbeit, sondern während des Studiums unterstützt hat. Ihr ständiger Einsatz, mich zu motivieren, war sicher ein großer Teil meines Erfolges an der FH Campus02.

Des Weiteren möchte ich mich bei meinem Betreuer, Herrn Prof. Dipl. Ing. Lutzmayr, für die Unterstützung und vor allem ehrlichen Worte, welche mich immer wieder gepusht haben, bedanken.

## **KURZFASSUNG**

In dieser Arbeit wird auf die Realisierung eines Soft- und Hardwarekonzeptes eingegangen, welches ein optisches Auswertesystem von Dartpfeilen auf einer Dartscheibe umfasst. Durch Literaturrecherche wurden Methoden und Ansätze zu optischen Bildverarbeitung erarbeitet.

Nach der Erstellung des Hardwarekonzepts, wird der mechanische Aufbau mit den ausgewählten Elektronikkomponenten durchgeführt. Mithilfe der Bildverarbeitungsbibliothek OpenCV wird in C# eine Testumgebung aufgesetzt, welche zur Findung der Einstellparameter und Erarbeitung des Auswertealgorithmus eingesetzt wird. Hierfür wird eine grafische Oberfläche erstellt, die eine benutzerfreundliche Bedienung aufweist. Der dadurch erzeugte Auswerte- und Positionierungsalgorithmus wird im nächsten Schritt in einer Software umgesetzt.

Die Resultate des Softwaretests zeigen, dass der Algorithmus funktioniert, jedoch noch Verbesserungspotenzial in Bezug auf die Genauigkeit aufweist. Zusätzlich muss ein Weg gefunden werden, die USB-BUS-Last zu minimieren beziehungsweise das simultane Einlesen von Kameras über einen USB-HUB zu vermeiden.

## **ABSTRACT**

This thesis deals with the realization of a software and hardware concept, which includes an optical evaluation system of darts on a dartboard. Through literature research, methods and approaches to optical image processing were developed.

After the creation of the hardware concept, the mechanical construction with the selected electronic components is carried out. With the help of the image processing library OpenCV a test environment is set up in C#, which is used for the determination of the adjustment parameters and the development of the evaluation algorithm. For this purpose, a user-friendly graphical interface is created. The resulting evaluation and positioning algorithm is being implemented in software in the next step.

The results of the software test show that the algorithm works, but still has room for improvement in terms of accuracy. In addition, a way has to be found to minimize the USB-BUS load, respectively to avoid the simultaneous reading of cameras via a USB-HUB.

## INHALTSVERZEICHNIS

1	Einleitung.....	1
1.1	Problemstellung .....	1
1.2	Zielsetzung.....	1
1.3	Systeme am Markt.....	2
1.3.1	Scolia System .....	2
1.3.2	OneHundredAndEightyCore .....	3
1.4	Fazit der Systeme am Markt.....	4
2	Einplatinencomputer.....	5
2.1	Definition eines Einplatinencomputer .....	7
2.2	Raspberry Pi .....	7
2.2.1	Übersicht der relevanten Raspberry Pi-Modelle.....	7
2.2.2	Vergleich der relevanten Raspberry Pi-Modelle .....	8
2.2.3	Hardwareaufbau eines Raspberry Pi.....	9
2.3	Odroid .....	10
2.3.1	Übersicht der relevanten Odroid-Modelle.....	11
2.3.2	Vergleich der relevanten ODROID-Modelle .....	12
2.3.3	Hardwareaufbau eines Odroid.....	12
3	Computer Vision.....	14
3.1	Einführung.....	14
3.2	Herausforderungen der Objekterkennung und Bildverarbeitung.....	14
3.2.1	Drehung, Spiegelung, Skalierung und Verzerrung .....	17
3.2.2	Lichteinfluss .....	18
3.2.3	Mehrere Objekte .....	19
3.3	Darstellung eines Bildes .....	19
3.3.1	RGB-Modell .....	19
3.3.2	Graustufenbild.....	20
3.3.3	Das Binärbild.....	21
3.4	Methoden zur Bildverarbeitung.....	21
3.4.1	Filter .....	21
3.4.1.1	Glättung .....	22
3.4.1.2	Die Filtermatrix .....	22
3.4.1.3	Anwendung eines Filters .....	23
3.4.1.4	Grenzen von Filter .....	24
3.4.2	Kantendetektion .....	24
4	OpenCV.....	27
4.1	Einführung in OCV .....	27
4.2	Stakeholders von OCV .....	27
4.3	Kontextübersicht von OCV .....	28
4.4	Programmiersprachen für OCV .....	30

4.5	OCV Strukturübersicht.....	30
4.6	Emgu CV.....	31
5	Anforderungsanalyse und Konzeptfindung .....	33
5.1	Grundidee .....	33
5.2	Anforderungsanalyse .....	33
5.2.1	Mechanische Anforderungsanalyse .....	33
5.2.2	Anforderungen an die Elektronikkomponenten .....	36
5.2.3	Softwareanforderungen .....	37
5.3	Gesamtkonzept.....	39
5.3.1	Konzept des Hardwareaufbaus .....	39
5.3.2	Softwarekonzept .....	40
5.3.2.1	Parameter- und Sucheinstellungen .....	41
5.3.2.2	Der Scan.....	42
5.3.2.3	Der Suchalgorithmus .....	42
6	Hardwareaufbau und Testumgebung.....	45
6.1	Hardwareaufbau und dessen Komponenten .....	45
6.2	Softwaretestumgebung.....	48
6.2.1	Einbinden von ECV in ein C#-Projekt.....	50
6.2.2	Einlesen des Originalbildes .....	51
6.2.3	Erstellung des GSB und BB.....	52
6.2.4	Filtereinstellungen für die Kantendetektion .....	53
6.2.5	Anzeigen eines Differenzbildes .....	55
6.2.6	Minimieren des Bildes auf eine ROI .....	56
6.2.7	Erzeugen eines Rechtecks zum Umspannen eines DP .....	57
6.2.8	Schnittpunkterrechnung der Symmetrielinie und Grundebene der DS .....	58
7	Umsetzung des Softwarealgorithmus .....	60
7.1	Kamerakalibrierung.....	60
7.2	Der Positionierungsalgorithmus.....	64
7.2.1	Trichterförmige Erfassung von Bildern von einer Kamera.....	64
7.2.2	Berechnung des Punktes eines Dartpfeils in einem Koordinatensystem .....	65
7.2.3	Punktebestimmung auf der Dartscheibe .....	66
7.2.4	Der Detektionsalgorithmus .....	68
7.3	Statistische Auswertung von erkannten DP .....	72
8	Fazit.....	75
8.1	Anforderungsanalyse und Konzepterstellung.....	75
8.2	Mechanischer Aufbau .....	75
8.3	Erarbeitung des Softwarealgorithmus .....	75
8.4	Erstellung der finalen Software und Softwaretest.....	76
8.5	Ausblick.....	76
	Literaturverzeichniss.....	77
	Abbildungsverzeichnis.....	80

Tabellenverzeichnis .....	83
Quelltextverzeichniss.....	84
Abkürzungsverzeichnis.....	85

# 1 EINLEITUNG

In den letzten Dekaden ist die Zahl offiziell eingetragener Dartspieler (DSP) auf über 500.000 gewachsen. Diese verteilen sich weltweit auf insgesamt 80 Länder.<sup>1</sup> Dadurch wächst auch die Zahl an aufgestellten Dartscheiben (DS) in privaten Haushalten. Mit der steigenden Zahl an DS und dem Fortschritt der Technik, wird die Nachfrage nach Auswertesystemen im Steeldart (SD) immer größer.

## 1.1 Problemstellung

Zurzeit gibt es mehrere Produkte am Markt, die eine automatisierte Punkteauswertung beim SD ermöglichen. Diese sind jedoch mit einem sehr großen finanziellen Aufwand verbunden und noch gibt es wenig brauchbare Alternativen für den Amateurspieler. Die verfügbaren „do it yourself“-Entwicklungen setzen meist ein technisches Hintergrundwissen sowie Eigeninitiative im Bereich der Informatik voraus.

Parallel zu dieser Entwicklung von Eigensystemen verändert sich auch das Hardwaresystem der jeweiligen Anwendung. So müssen komplexe Bildverarbeitungsalgorithmen nicht mehr auf leistungsstarken Standrechnern abgearbeitet werden, sondern können mittlerweile auf Einplatinencomputern kompakt und kostengünstig umgesetzt werden. Auf diesen Einplatinencomputern können mittlerweile fast alle Programmier- und Skriptsprachen ausgeführt werden.

Mit der Kenntnis dieser Informationen kann nun an einer Lösung gearbeitet werden, die auf die kostengünstige Umsetzung eines Auswertesystems für SD abzielt. Zusätzlich soll die Arbeit ein Leitfadensystem für die nicht technisch versierten Anwender darstellen, der leicht und mit wenig Aufwand umzusetzen ist.

## 1.2 Zielsetzung

Das fertige System soll die Anforderungen, welche zu Beginn des praktischen Teils in einer Anforderungsanalyse erarbeitet werden, erfüllen, und die Software auf kommerziellen Systemen und Einplatinenrechnern ausführbar sein. Durch Literaturrecherche und praktische Experimente werden unterschiedliche Bildverarbeitungsbibliotheken und deren Funktionen analysiert und Ziel ist es, einen geeigneten Algorithmus zu finden, der ohne Verletzung von Schutzrechten und ohne die Verwendung von kostenpflichtigen Bibliotheken auskommt. Darüber hinaus muss dabei der Materialeinsatz für den Endbenutzer möglichst klein und leicht zugänglich gehalten werden. Eine Dokumentation der Software sowie eine Aufbauanleitung mit Materialliste soll erstellt werden. Darüber hinaus wird sowohl eine geeignete Programmiersprache als auch eine passende Bildverarbeitungsbibliothek ausgewählt und der Algorithmus aufgesetzt, wobei die Software mit handelsüblicher Hardware einsatzfähig sein sollte. Eine Aufbaudokumentation wird erstellt und die Software muss anschließend mittels Testumgebung, Softwaretest und Feldversuch validiert und die Ergebnisse aufgezeichnet werden.

---

<sup>1</sup> Vgl. Krist (2021), Online-Quelle [21.05.2022].

## 1.3 Systeme am Markt

In dem folgenden Kapitel wird eine Übersicht über die aktuell populärsten Systeme am Markt dargestellt. Unterschiede sowie Vor- und Nachteile werden aufgezeigt und gegenübergestellt. Zusätzlich wird auf die Hard- und Software des jeweiligen Systems eingegangen

### 1.3.1 Scolia System

Das Scolia System (SCS) ist das einzige käufliche System am Markt und ist von dem ungarischen Unternehmen Scolia Technologies Ltd. Entwickelt worden. Das SCS wirbt vor allem mit der Genauigkeit, die laut eigenen Angaben bei >99 % liegt, und einer simplen Installation ihrer Systeme.<sup>2</sup>

Die folgende Tabelle zeigt Informationen über das SCS:

Anzahl der Kameras:	3 Stück mit 720p Auflösung
Auswertesystem:	Odroid C4
Programmiersprache:	C++
Preis:	
Scolia Home:	>700 €
Scolia Pro:	1300 €

Tabelle 1: Informationen SCS, Quelle: Eigenen Darstellung.

Der Aufbau des SCS besteht aus 3 Kameras die schrag auf die DS gerichtet sind. In der folgenden Abbildung 1 wird der Aufbau dargestellt. Der Algorithmus dahinter ist nicht einsehbar und es kann nur vermutet werden, wie dieser umgesetzt worden ist.

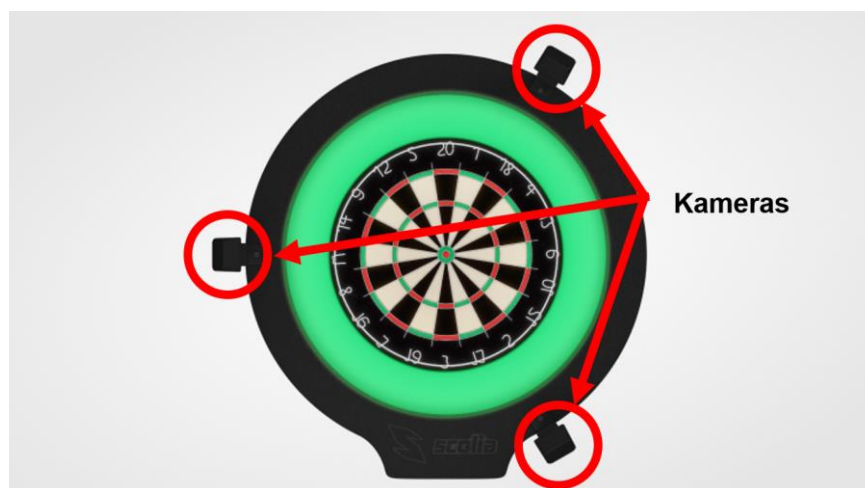


Abbildung 1: Aufbau Scolia System, Quelle: Scolia Technologies Ltd. (2020), Online-Quelle [21.05.2022].

<sup>2</sup> Vgl. Scolia Technologies Ltd. (2020), Online-Quelle [21.05.2022].



Das SCS bietet einige Features, die ständig verbessert werden, und es ist damit ein konventionelles und professionelles Produkt. Diese Features sind wie folgt:

- Lokales Spiel mit mehreren Spielern
- Onlinespiele
- Computer-Gegner
- Analytische Auswertungen und Statistik
- Board Monitoring
- Mastercaller, Soundausgabe

Mit Anschaffungskosten von über 700 € bietet das SCS ein sehr ausgereiftes System, welches mit Onlineupdates ausgestattet ist.<sup>3</sup>

### 1.3.2 OneHundredAndEightyCore

OneHundredAndEightyCore (OHEC) ist ein Open Source-Projekt und ist von Eugene Boldrow entwickelt worden. Mittlerweile ist es ein Community-Projekt, mit dem Namen YelloFive, welches von mehr als 10 Mitgliedern weiterentwickelt wurde. Das letzte bekannte Update ist im Oktober 2020 erfolgt und danach nicht mehr weiterentwickelt worden.<sup>4</sup>

Anzahl der Kameras	4 Stück mit mindestens 360p Auflösung
Auswertesystem	Standrechner/Laptop
Programmiersprache	C++
Preis	Freeware

Tabelle 2: Informationen OHEC, Quelle: Eigene Darstellung.

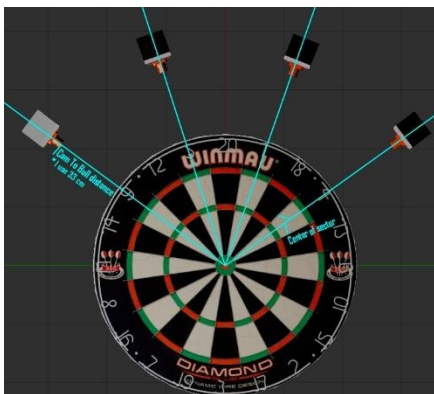


Abbildung 2: OHEC System, schematischer Aufbau, Quelle: YellowFive5 (2020), Online-Quelle [21.05.2022].

OHEC ist ein 4-Kamerasystem, wobei die Kameras hierbei seitlich auf der Dartscheibe angebracht werden und sich beinahe auf gleicher Ebene mit derselben befinden. Die Auswertung erfolgt mittels Auswertung des Koordinatensystems. In der folgenden Abbildung wird der theoretische Aufbau des Systems dargestellt. Da die Entwickler darauf Wert gelegt haben, dass der Benutzer sich den Aufbau selbst erarbeitet, gibt es zu diesem System nur schematische Bilder. Hier wird auf die Flexibilität der Benutzer gebaut, was einen sehr hohen Grad an technischem Knowhow voraussetzt. Die Entwicklung ist jedoch Ende 2020 eingestellt und nicht mehr weiterentwickelt worden.

---

<sup>3</sup> Vgl. Scolia Technologies Ltd. (2020), Online-Quelle [21.05.2022].

<sup>4</sup> Vgl. Boldrow (2020), Online-Quelle [21.05.2022].

## **1.4 Fazit der Systeme am Markt**

OHEC weist gute Ansätze für die Auswertung von DP auf einer DS auf, jedoch ist das System bis dato nie zu Ende entwickelt worden. Zusätzlich zeigt das System große Schwächen in der Modularität, Genauigkeit und Verständlichkeit auf. Der hohe Grad an Informatikverständnis, der für dieses System vorausgesetzt wird, macht es für einen Laien eher uninteressant.

Das SCS ist ein hochentwickeltes System, welches auf dem Markt käuflich erworben werden kann. Der Preis ist zwar gerechtfertigt, für den Privatgebrauch jedoch sehr hoch.

Die Motivation, ein kostenfreies System zu entwickeln, ist dadurch weiterhin gegeben. Das Ziel dieser Arbeit ist, bestehende Ideen und Konzepte einfließen zu lassen, um ein neues System zu entwickeln.

## 2 EINPLATINENCOMPUTER

In dem folgenden Kapitel wird auf die unterschiedlichen Einplatinencomputer (EPC) eingegangen. Da die EPC mittlerweile eine hohe Leistung aufweisen, ist es möglich, geeignete Auswertesysteme darauf auszuführen. Die richtige Balance zwischen Leistung und Preis spielt hier eine enorm große Rolle und unnötige Ressourcen sollen weitestgehend vermieden werden. Durch die gezielte Auswahl kann so eine Basis geschaffen werden, die kompakt und universell einsetzbar ist. In diesem Kapitel wird übersichtsmäßig eine große Palette von EPC dargestellt, jedoch nur auf die kommerziellen und weit verbreiteten Systeme eingegangen. Zu Beginn werden Begriffe erklärt, um ein grundlegendes Verständnis zu schaffen.

		Multimedia									
			Odroid-N2+		LattePanda Delta				Odroid-H2+		
	Wand-board IMX8M-Plus			Odroid-C4		Rock Pi 4C	Odroid-HC4				
	Tinker Edge T	Vizi-AI				Rock Pi X			Odyssey-X86		
	Coral Dev Board Mini				PineCube				ROC-RK3308-CC Plus		
<b>AI</b>	Humming Board Mate			Humming Board Ripple			I-Pi				<b>Networking</b>
				MaxBoard Mini					Orange Pi Zero2		
				Lindenis V536		BPI-F2P	BB Enhanced	BB Green Gateway			
				Odyssey-STM	Quantum Mini		Rock Pi E	Orange Pi R1 Plus	Stinger96		
				Sipeed TANG Hex		NanoPi Neo3	NanoPi R4S	NanoPi R2S	BPI-R64		
											<b>IoT</b>

Abbildung 3: Feature-Matrix Tabelle von EPC, Quelle: Brown (2021), Online-Quelle [21.05.2022].

Abbildung 3 zeigt die drei größten Bereiche für den Einsatz von EPC im Jahr 2020: Netzwerk, KI (Künstliche Intelligenz) und Multimedia. Es besteht eine beträchtliche Nachfrage nach überwiegend kopflosen Dual-LAN-, zunehmend auch Dual-Gigabit-Ethernet-Netzwerk- und IoT (Interne of Things)-Boards.<sup>5</sup>

<sup>5</sup> Vgl. Brown (2021), Online-Quelle [21.05.2022].

Wie bei den meisten elektronischen Komponenten gibt es beim preislichen Aspekt eine Spanne, die nach oben kaum Grenzen kennt. Die folgende Abbildung 5 soll einen Überblick über die handelsüblichen EPC und ihre Preis-Leistung-Klassifizierung geben.

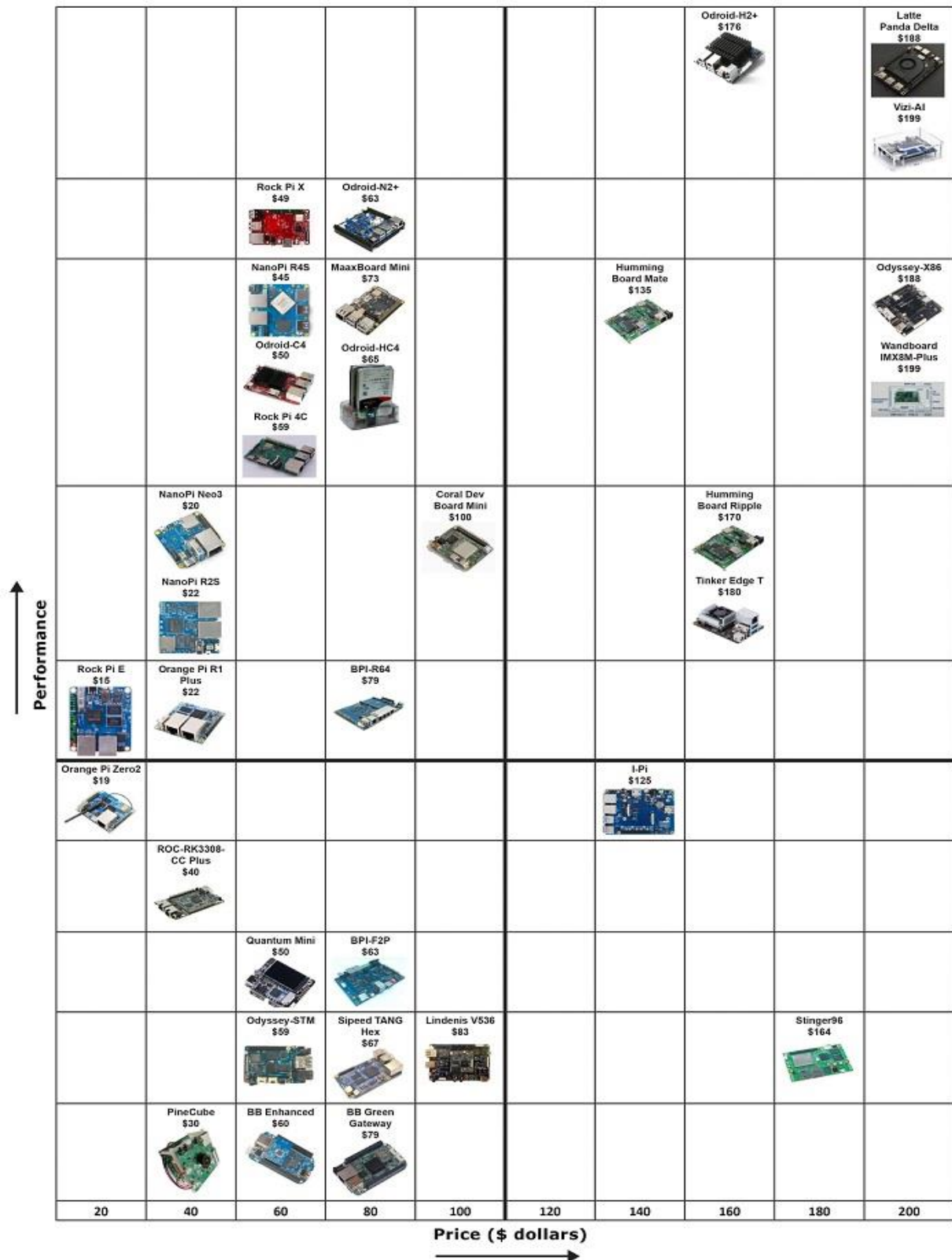


Abbildung 4: Preis-Leistung-Darstellung von EPC, Quelle: Brown (2021), Online-Quelle [21.05.2022].

## 2.1 Definition eines Einplatinencomputer

Ein EPC oder Single-Board-Computer ist ein Computer, der auf einer einzigen Leiterplatte aufgebaut ist. Er ist mit Mikroprozessor(en), Speicher, Ein-/Ausgabeschnittstellen und anderen Merkmalen ausgestattet, die für einen funktionsfähigen Computer erforderlich sind. EPC bieten in der Regel eine lüfterlose Architektur mit geringem Stromverbrauch. Aktuelle EPC sind mit einer Vielzahl von Prozessortypen ausgestattet, die meisten mit integrierten GPUs. Diese Prozessoren reichen von X86-basierten Prozessoren aus dem traditionellen PC-Bereich (AMD und Intel) bis hin zu ARM-Prozessoren, die traditionell in der Industrie und neuerdings auch im Mobilbereich verwendet werden. Die am weitesten verbreitete Software, die auf EPC verwendet wird, ist Linux mit zahlreichen Ableitungen, darunter Android, Ubuntu, Fedora, Debian und Arch Linux sowie FreeBSD. Viele der heutigen EPC sind so leistungsfähig geworden, dass sie allmählich die Fähigkeiten moderner PCs und Tablets besitzen. Dieser Trend wird sich fortsetzen, da leistungsstärkere Prozessoren ihren Weg in den Embedded-Computing-Markt finden, da sich das Preis-Leistungs-Verhältnis ständig verbessert und zusätzliche Hersteller in diese Branche einsteigen.<sup>6</sup>

## 2.2 Raspberry Pi

Der Raspberry Pi (RP) ist der bekannteste EPC auf dem Markt. Seine Größe entspricht dem einer Kreditkarte beziehungsweise eines Smartphones und er besitzt typische Anschlüsse wie ein Computer, zum Beispiel USB-Anschlüsse. Je nach Bauart und Version des RP besitzt er auch eine unterschiedliche Anzahl von GPIOs (General Purpose Input/Output)-Pins, welche in den Bereich des Microcontrollers geht.<sup>7</sup> Als der RP auf dem Markt kommt, feiert er sofort einen Erfolg und schnell mit den Verkaufszahlen durch die Decke. Die Anfangs 10.000 Stück geplanten Einheiten sind sofort übertroffen und zwei Millionen Stück in den ersten zwei Jahren verkauft worden. Bis 2016 hat die britische Raspberry Pi Foundation über acht Millionen Exemplare von unterschiedlichen RP Modellen verkauft.<sup>8</sup>

Ein großer Aspekt der Beliebtheit ist die Open-Source-Community. Da der RP auf LINUX basiert, welches ein freies Betriebssystem (BS) ist, ist es leicht mit Android-Geräten sowie zahlreichen Internet-Servern kompatibel.<sup>9</sup> Dadurch findet man auch als Einsteiger sofort Hilfestellungen im Internet und in Foren.

### 2.2.1 Übersicht der relevanten Raspberry Pi-Modelle

Um einen Überblick über die unterschiedlichen Modelle des RP zu bekommen, werden die wichtigsten kurz beschrieben und dargestellt.

---

<sup>6</sup> Vgl. Mulyana (2016), Online-Quelle [21.05.2022].

<sup>7</sup> Vgl. Kofler/Charly/Scherbeck (2016), S. 21f.

<sup>8</sup> Vgl. Kofler/Charly/Scherbeck (2016), S. 15.

<sup>9</sup> Vgl. Kofler/Charly/Scherbeck (2016), S. 16.

### **Raspberry Pi Modell A und B**

Mit einem Preis von 35 € sind die ersten Modelle des RP ein günstiger Einstieg in die Welt der EPC. Sie besitzen einen Single-Core-Prozessor mit 700 MHz und unterscheiden sich in der Größe des Arbeitsspeichers. Während das Modell A 256 Mbyte aufweist, verfügt das Modell B mit 512 Mbyte über die doppelte Arbeitsspeicherkapazität. Damit ist das Modell B im Gegensatz zu Modell A eher für grafische Anwendung geeignet.<sup>10</sup>

### **Raspberry Pi Modell 2**

Mit 40 € ist die Weiterentwicklung des RPs Modell A und B noch immer ein sehr günstiger EPC. Der Pi 2 ist schon mit einem 900 MHz Quad-Core-Prozessor ausgestattet. Zusätzlich besitzt er eine Dual-Core-GPU (Graphics-Processing-Unit) und einen Arbeitsspeicher mit 1 GByte SDRAM (Synchronous Dynamic Random Access Memory). Der wesentliche Unterschied zum Vorgänger besteht jedoch in der ARMv7-Architektur, welche die Möglichkeit bietet, unterschiedliche BS, wie Ubuntu oder spezielle Versionen von Windows, zu nutzen.<sup>11</sup>

### **Raspberry Pi Model 3 B**

Mit einem Preis von 45 € ist das Model Pi 3 B noch immer ein kostengünstiger EPC. Mit seinen mittlerweile 1,2GHz Quad-Core-Prozessor ist er um 30 Prozent leistungsfähiger als sein Vorgänger. Dank der immer gleichbleibenden Baugröße (86 x 56 x 20 cm) ist er in allen Gehäusen der Vorgängermodelle einbaubar. Die Arbeitsspeichergröße bleibt wie im Vorgänger bei 1 GByte SDRAM. Die externe SD-Karte von durch eine microSD-Karte ersetzt. Zusätzlich ist in der dritten Generation noch eine WLAN-Karte nach IEEE 802 und eine Low Energy Bluetooth-Schnittstelle hinzugefügt worden. Eine weitere Entwicklung der dritten Generation ist die 64-Bit-Unterstützung, was jedoch auch bedeutet, dass das BS auch 64-bit unterstützen muss.<sup>12</sup>

### **Raspberry Pi Modell 4 B**

Das Pi 4 B-Modell ist seit 2019 die aktuell neueste Baureihe der RP Familie. Mit zwei USB 3.0 Ports lassen sich viel größere Daten übertragen als noch bei den Vorgängermodellen. Er ist mit einem 1,5G Hz Quad-Core-Prozessor ausgestattet und daher noch leistungsstärker als seine Vorgänger. Der Pi 4 B ist mit 1, 2, 4 oder 8 GByte-Arbeitsspeicher verfügbar und besitzt mittlerweile LPDDR4-SDRAM. Das LP steht hier für Low-Power und ist somit die Energiespar-Variante.<sup>13</sup>

## **2.2.2 Vergleich der relevanten Raspberry Pi-Modelle**

In der folgenden Tabelle 3 sollen die einzelnen RP-Modelle gegenübergestellt werden, wobei nur die wichtigsten Merkmale berücksichtigt werden:

---

<sup>10</sup> Vgl. Schnabel (2022), Online-Quelle [21.05.2022].

<sup>11</sup> Vgl. Schnabel (2022), Online-Quelle [21.05.2022].

<sup>12</sup> Vgl. Schnabel (2022), Online-Quelle [21.05.2022].

<sup>13</sup> Vgl. Schnabel (2022), Online-Quelle [21.05.2022].

	<b>Modell A und B</b>	<b>Modell 2</b>	<b>Modell 3 B</b>	<b>Modell 4 B</b>
<b>CPU</b>	Broadcom BCM2835 Single- Core ARMv6, 700MHz	Broadcom BCM2836 Quad- Core ARMv7 900MHz	Broadcom BCM2837 Quad- Core ARMv8, 1,2 GHz 64-bit	Broadcom BCM2711 Quad- Core Cortex-A72, 1,5 GHz, 64-bit
<b>RAM</b>	256 Mbyte, 512 Mbyte SDRAM	1 GB, LPDDR2- SDRAM	1 GB, LPDDR2- SDRAM	1 GB, 2 GB, 4 GB, 8 GB  LPDDR4-SDRAM
<b>Video</b>	H.264, OpenGL	H.264, OpenGL	H.264, OpenGL	H.265, H.264, OpenGL
<b>WLAN</b>	Kein Anschluss	Kein Anschluss	IEEE 802.11b,g,n (2,4 GHz)	IEEE 802.11 b/g/n/ac (2,4 und 5 GHz)
<b>Bluetooth</b>	Kein Anschluss	Kein Anschluss	4.1	5.0
<b>USB</b>	USB 2.0 (2 Ports)	USB 2.0 (4 Ports)	USB 2.0 (4 Ports)	USB 2.0, USB 3.0 (je 2 Ports)
<b>LAN</b>	10/100 Mbit/s Ethernet RJ45 Port	10/100 Mbit/s Ethernet RJ45 Port	10/100 MBit/s Ethernet RJ45 Port	Gigabit Ethernet RJ45 Port

Tabelle 3: Vergleich der Raspberry Pi-Modelle, Quelle: Eigene Darstellung.

### 2.2.3 Hardwareaufbau eines Raspberry Pi

Um einen Überblick über den Hardwareaufbau des RP zu bekommen, werden in der in der folgenden Abbildung 5 die Hauptkomponenten dargestellt:

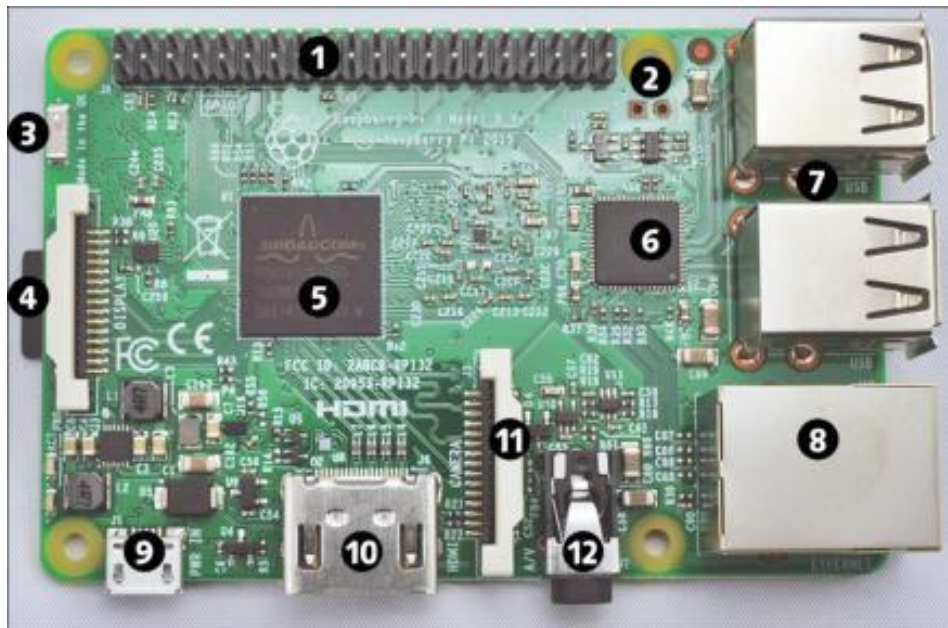


Abbildung 5: RP 3 B Hardwareaufbau, Quelle: Kofler/Charly/Scherbeck (2016) (leicht modifiziert), S. 23.

- |  |                                    |
|--|------------------------------------|
| 1: GPIO-Header J8 40-polig               | 7: 2x 2.0 USB-Ports                |
| 2: Run-Header (P6)                       | 8: RJ45 LAN-Anschluss              |
| 3: Keramikantenne für WLAN und Bluetooth | 9: Micro-USB-Buchse 5V (DC Supply) |
| 4: DSI Anschluss für Display             | 10: HDMI-Anschluss                 |
| 5: SoC BCM2837                           | 11: CSI-Anschluss für Kamera Board |
| 6: LAN-/USB-Controller                   | 12: 3,5mm Audio- und Videoausgang  |

## 2.3 Odroid

Odroid (OD) steht für Open-Droid und ist eine Entwicklungsplattform für Hard- und Software.<sup>14</sup> Er ist neben dem RP einer der meist benutzten EPC am Markt und es existieren ungefähr 15 Typen. Der aus Südkorea stammende OD gehört zu den größten Konkurrenten von RP und startete schon im Jahr 2011 mit dem ersten EPC. Die Strategie des OD im Gegensatz zum RP ist, durch etwas teurere Hardware deutlich höhere Leistung anzubieten. Die schnell wachsende Variantenvielfalt innerhalb von kürzester Zeit erschwert eine gute Übersicht über die Unterschiede der einzelnen Bauarten. Es ist jedoch schnell zum Auslaufen einiger OD-Projekte gekommen, da sich die Firma neu strukturiert und einige obsoletere Modelle eingestellt hat.<sup>15</sup>

<sup>14</sup> Vgl. odroid (2017), Online-Quelle [21.05.2022].

<sup>15</sup> Vgl. Löbbeling (2022), Online-Quelle [21.05.2022].



### 2.3.1 Übersicht der relevanten Odroid-Modelle

Um einen Überblick über die unterschiedlichen Modelle des OD zu bekommen, sollen die wichtigsten aufgelistet und beschrieben:

#### **Odroid C1**

Der OD C1 ist erstmals 2015 auf den Markt gekommen.<sup>16</sup> Er ist mit 45 € einer der leistungsstärksten EPC in seiner Preisklasse. Mit 1.5 GHz ist er leistungsstärker als der RP Modell A und besitzt auch schon einen Quad-Core Prozessor. Zusätzlich besitzt er auch schon eine Gigabit Ethernet-Schnittstelle. Da der OD C1 schon mit der ARMv7-Technologie ausgestattet ist, lassen sich hier schon unterschiedliche BS bedienen. Hauptsächlich werden UBUNTU, Android 4.4 und andere Linux OS betrieben.<sup>17</sup>

#### **Odroid C2**

Das OD C2 ist eine schrittweise Verbesserung des C1+. Wie der RP 3 verwendet der C2 einen ARM Cortex-A53 Quad-Core-Prozessor. Im Gegensatz zum RP 3 verfügt der OD C2 über 2 GB RAM und HDMI 2.0, das 4 k-Videos bei 60 Hz unterstützt.<sup>18</sup> In Bezug auf die Prozessortaktfrequenz gibt es hier keine Unterschiede zum Vorgänger, jedoch besitzt der OD C2 schon die 64bit-Technologie. Bei diesem Modell befindet man sich mit 90 € schon in einer höheren Preisklasse.

#### **ODROID C4**

Der OD-C4 gehört zu den EPC der neuen Generation, der energieeffizienter und leistungsfähiger ist als OD C2. Der Hauptprozessor des OD C4 ist Quad-Core-Cortex-A55 mit einer GPU Mali-G31 der neuen Generation. Die A55-Kerne laufen mit 2,0 GHz ohne thermische Drosselung unter Verwendung des serienmäßigen Kühlkörpers, was einen robusten und leisen Computer ermöglicht. Die CPU-Multi-Core-Leistung ist etwa 40 % schneller und die System-DRAM-Leistung ist 50 % schneller als beim ODROID-C2.<sup>19</sup> Ein Unterschied zu RP ist der fehlende WLAN-Chip. Dieser kann mittels nachrüstbarem WLAN-USB Stick ergänzt werden.

---

<sup>16</sup> Vgl. Online (2022), Online-Quelle [21.05.2022].

<sup>17</sup> Vgl. wiki.ordoid.com (2020), Online-Quelle [21.05.2022].

<sup>18</sup> Vgl. MUO (2022), Online-Quelle [21.05.2022].

<sup>19</sup> Vgl. wiki.ordoid.com (2020), Online-Quelle [21.05.2022].

### 2.3.2 Vergleich der relevanten ODROID-Modelle

In der folgenden Tabelle ist eine Gegenüberstellung der einzelnen OD-Modelle vorgenommen worden, wobei auch hier wieder nur die wichtigsten Merkmale berücksichtigt werden:

	<b>Modell C1</b>	<b>Modell C2</b>	<b>Modell C4</b>
<b>CPU</b>	Amlogic S805 ARM Cortex-A5(ARMv7) Quad- Core 1.5 Ghz	Amlogic S905 ARM Cortex-A53(ARMv8) Quad-Core 1.5 Ghz	Amlogic S905X3 ARM Cortex-A55(ARMv8) Quad-Core 2.0 Ghz
<b>RAM</b>	1 GB DDR3-SDRAM	2 GB, DDR3-SDRAM	4 GB, DDR4-SDRAM
<b>Video</b>	H.264, OpenGL	H.264 4 k 60/30 FPS	H.265 4 k 60 FPS, H.264 4 k 30 FPS
<b>WLAN</b>	Kein Anschluss	Kein Anschluss	Kein Anschluss
<b>Bluetooth</b>	Kein Anschluss	Kein Anschluss	Kein Anschluss
<b>USB</b>	USB 2.0 (4 Ports)	USB 2.0 (4 Ports)	USB 3.0 (4 Ports)
<b>LAN</b>	10/100/1000 Mbit/s Ethernet RJ45 Port	10/100/1000 Mbit/s Ethernet RJ45 Port	10/100/1000 MBit/s Ethernet RJ45 Port

Tabelle 4: Vergleich der Odroid-Modelle, Quelle: Eigene Darstellung.

### 2.3.3 Hardwareaufbau eines Odroid

Um einen Überblick über den Hardwareaufbau des OD zu bekommen, werden in der in der folgenden Abbildung 6 die Hauptkomponenten dargestellt:

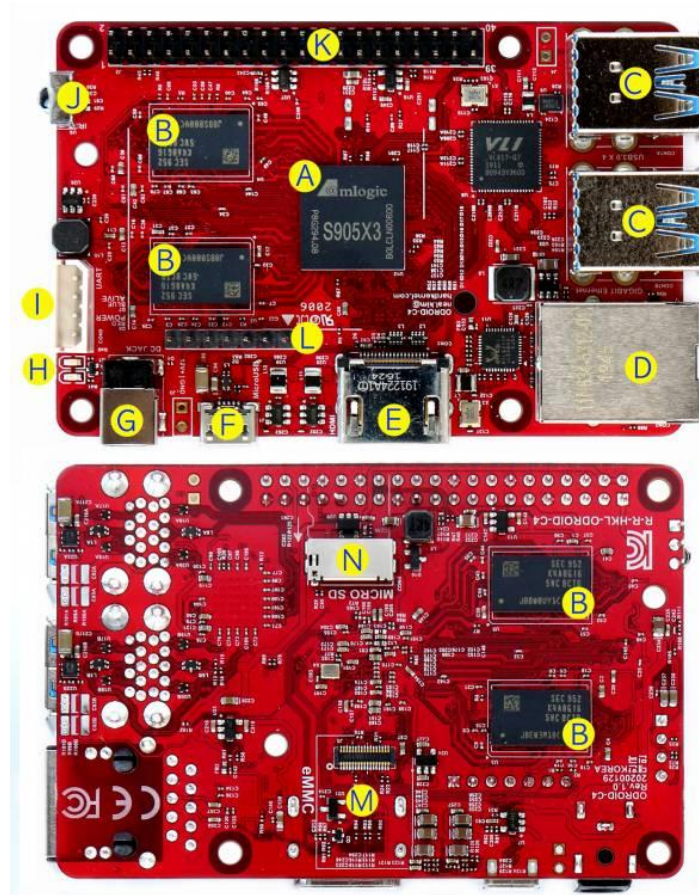


Abbildung 6: OD C4 Hardwareaufbau, Quelle: wiki.ordoid.com (2020), Online-Quelle [21.05.2022].

- |                          |  |
|--------------------------|--|
| A: CPU (Armlogic S905X3) | H: 2x System LED (Statusindikatoren)   |
| B: DDR4 RAM (4 GiB)      | I: UART (Universal Asynchronous Rx/Tx) |
| C: 2x 3.0 USB-Ports      | J: Infrarotschnittstelle               |
| D: RJ45 LAN-Anschluss    | K: 40x GPIO                            |
| E: HDMI-Anschluss        | L: 7x GPIO                             |
| F: Micro-USB 2.0 Port    | M: eMMC-Modul (Flashspeicher)          |
| G: DC-Power Buchse       | N: Micro SD-Slot                       |

### 3 COMPUTER VISION

In dem folgenden Kapitel wird auf das Teilgebiet Computer Vision (CV) eingegangen. Eine grundlegende Definition und Herausforderungen in CV werden vorgenommen. Zusätzlich werden Methoden zur Objekterkennung und Bildverarbeitung aufgelistet.

#### 3.1 Einführung

CV ist ein Teilbereich der Informatik, welcher es dem Computer und Systemen ermöglicht, digitale Bilder, Videos und Szenen-Informationen abzuleiten und aus diesen Informationen abzuleiten.<sup>20</sup>

Die drei Grundgedanken von CV sind:

- Bilder sehen
- Bilder verstehen
- Bilder analysieren

CV funktioniert ähnlich wie das menschliche Sehen. Jedoch wird der Mensch lebenslang trainiert, um Objekte zu erkennen und diese zu unterscheiden. Bei CV werden Maschinen, Computer und Systeme darauf trainiert, diese komplexen Aufgaben zu erlernen.<sup>21</sup>

#### 3.2 Herausforderungen der Objekterkennung und Bildverarbeitung

In dem folgenden Kapitel werden die Herausforderungen der Bildverarbeitung und Objekterkennung dargestellt. Die Schwierigkeiten der Objekterkennung sind umfangreich. Formen können sich aus verschiedenen Gründen in ihrem Aussehen unterscheiden. Der wichtigste Grund hierfür sind die unterschiedlichen Perspektiven, die wir auf eine Form haben können. Formen können aus verschiedenen Winkeln und Positionen betrachtet werden, wodurch die Form größer, auf dem Kopf stehend, gekippt usw. erscheinen kann. Wir haben es aber auch mit Formen zu tun, die zwar dasselbe Konzept darstellen, in Wirklichkeit aber unterschiedlich sind. Keine zwei Bäume werden jemals genau gleich aussehen.<sup>22</sup>

Bei CV geht es darüber hinaus darum, von eingelesenen Daten, aus einem Einzelbild oder einer Videodatei, eine Darstellung zu generieren und/oder eine Entscheidung zu treffen. Dies führt zu einem vordefinierten Ziel. Zum Beispiel könnte eine Kamera auf einem Auto montiert sein, die kontextbezogene Informationen liefert, oder ein Entfernungsmesser zeigt an, dass ein Objekt einen Meter entfernt ist. Eine Entscheidung könnte lauten: „Es befindet sich eine Person in dieser Szene“ oder: „Es befinden sich 14 Tumorzellen auf dem Objektträger“. Eine Darstellung einer Szene kann zum Beispiel ein Farbbild in ein Bild mit Graustufen umwandeln oder Kamerabewegungen aus einer Sequenz entfernen.<sup>23</sup>

---

<sup>20</sup> Vgl. Marr (2019), Online-Quelle [21.05.2022].

<sup>21</sup> Vgl. IBM (2022), Online-Quelle [21.05.2022].

<sup>22</sup> Vgl. Object Recognition (2004), S. 2.

<sup>23</sup> Vgl. Bradski/Kaebler (2008). S. 2.

Im Gegensatz zum menschlichen Sehen erhält der Computer oder die Maschine nur ein Zahlenraster aus einer Bilddatei oder einer Kamera. Das menschliche Gehirn verarbeitet Gesehenes umgehend und kann daraus sofort Entscheidungen und Informationen ableiten. In den meisten Fällen gibt es bei CV keine eingebaute Mustererkennung sowie Steuerung des Fokus, was beim menschlichen Sehen automatisch passiert. Dies erfolgt durch jahrelanges Training des menschlichen Gehirns.<sup>24</sup> Dies führt zu einem weiteren Problem, das CV vor Herausforderungen stellt, nämlich das Verstehen von Bildern.

In CV gibt es mehrere Problemtypen, die von der Bildklassifizierung, Zuordnung einer Kategorie in einem Bild, bis hin zur semantischen Segmentierung und der Bestimmung einer Gruppe von Pixeln reichen.<sup>25</sup>

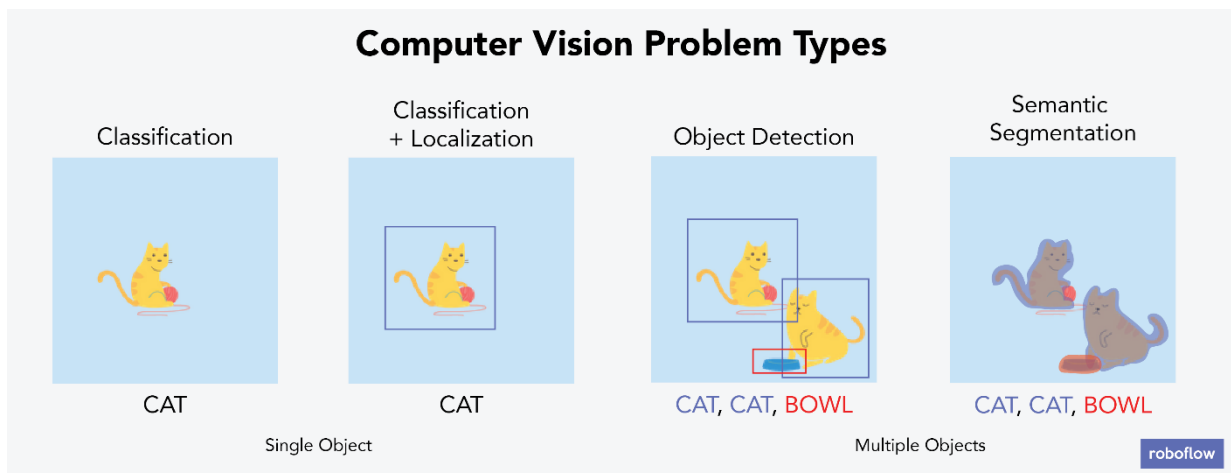


Abbildung 7: Probleme der CV, Quelle: Nelson (2020), Online-Quelle [21.05.2022].

Die Abbildung 7 zeigt ein simples Beispiel unterschiedlicher Aufgaben und Herausforderungen der CV. Zum einfacheren Verständnis werden die einzelnen Aufgaben im Folgenden definiert.

### **Klassifizierung (Classification):**

Im Allgemeinen wird unter Klassifizierung das Einteilen von Wissen und Informationen nach einheitlichen Prinzipien verstanden. Dies bedeutet, man weist jemanden oder etwas zu einer gewissen Gruppe zu.<sup>26</sup> Wie in der Abbildung 7 zu sehen ist, wird im ersten Bildausschnitt die Information extrahiert und das gesehene Objekt zu der Tierart Katze hinzugefügt.

### **Klassifizierung und Positionierung (Classification and Location):**

Im zweiten Bildausschnitt wird zusätzlich zur Klassifizierung der Information noch die Position der Klasse definiert. Hier wird ein Begrenzungsrahmen um die Katze gezogen und dieser mittels Höhen- und Breitenpixels positioniert<sup>27</sup>. Dadurch kann die Position des Rahmens im Bild bestimmt werden.

<sup>24</sup> Vgl. Danner (2020), Online-Quelle [21.05.2022].

<sup>25</sup> Vgl. Nelson (2020), Online-Quelle [21.05.2022].

<sup>26</sup> Vgl. Jörg/Uszkoreit (2015), Online-Quelle [21.05.2022].

<sup>27</sup> Vgl. Nelson (2020), Online-Quelle [21.05.2022].

**Objekterkennung (Object Detection):**

Bei Objekterkennungsproblemen geht es darum, Begrenzungsrahmen einzufügen, um Pixel von Interesse zu identifizieren und die Pixel in diesem Rahmen einer Klasse zuzuordnen. Bei Objekterkennungsproblemen können wir ein einzelnes Bild nach mehreren Boxen verschiedener Klassen durchsuchen.<sup>28</sup> In der Abbildung wird farblich dargestellt, wie die Begrenzungsrahmen unterschiedliche Objekte umschließen und diese zu unterschiedlichen eindeutigen Klassen zuordnen.

**Semantischer Segmentierung (Semantic Segmentation):**

Die Semantik ist wie folgt definiert: „Teilgebiet der Linguistik, das sich mit den Bedeutungen sprachlicher Zeichen und Zeichenfolgen befasst.“<sup>29</sup>

Semantische Segmentierungsprobleme, wie in der Abbildung 7 dargestellt, zielen darauf ab, die Umrisse von Pixeln zu isolieren. Dies geschieht bei jeder möglichen Form. Zusätzlich wird das isolierte Objekt von Interesse beschrieben.<sup>30</sup>

Eine weitere Herausforderung der CV stellt die Darstellung eines Bildes dar. Wird dem Computer ein Bild von einer Kamera oder Bilddatei zur Darstellung geschickt, erkennt dieser die Quelldatei als Zahlenraster. Dies hängt von der Auflösung des Bildformats sowie der farblichen Darstellung ab. So sieht zum Beispiel ein Mensch im Rückspiegel eines Autos die genauen Geschehnisse, eine montierte Kamera würde diese jedoch nur als Zahlenwert erkenne.<sup>31</sup>

In der folgenden Abbildung 8 wird dieser Fall dargestellt:

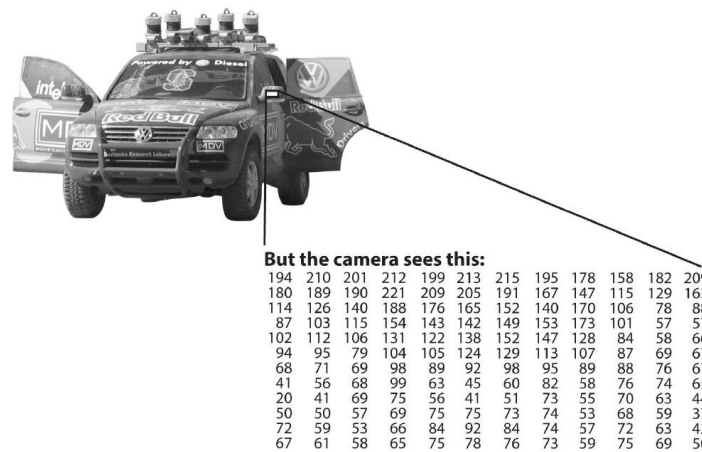


Abbildung 8: Montierte Kamera auf einem Rückspiegel: Quelle: Bradski/Kaebler (2008), S. 4 (leicht modifiziert).

<sup>28</sup> Vgl. Nelson (2020), Online-Quelle [21.05.2022].

<sup>29</sup> Bibliographisches Institut GmbH (2022), Online-Quelle [21.05.2022].

<sup>30</sup> Vgl. Nelson (2020), Online-Quelle [21.05.2022].

<sup>31</sup> Vgl. Bradski/Kaebler (2008), S. 3f.

Darüber hinaus werden Bilder nie mit den gleichen Voraussetzungen von einer Kamera aufgenommen. Einflüsse wie Licht, Reflexionen, Wetter und Bewegungen machen die Reproduktion eines Bildes unmöglich.<sup>32</sup> Außerdem kann ein Bild von mehreren Perspektiven aufgenommen werden, die jedoch das gleiche Objekt darstellen. Die Abbildung 9 zeigt den Eiffelturm von mehreren Perspektiven.



Abbildung 9: Darstellung des Eiffelturms von mehreren Perspektiven, Quelle: Gollapudi (2019), S. 26.

Der Umgang mit all diesen Fällen macht es der CV schwer, Probleme zu lösen. Beim Menschen erfolgt die Sammlung von Daten (durch Sehen) ständig. Anders als bei Maschinen ist die Wahrscheinlichkeit gering, dass ein Mensch einen Hund falsch einordnet, wenn er ihn in verschiedenen Positionen sieht.<sup>33</sup>

### 3.2.1 Drehung, Spiegelung, Skalierung und Verzerrung

Ein grundlegendes Problem bei der Erkennung einer Form ist die Tatsache, dass sie in jeder beliebigen Position gedreht, gespiegelt etc. werden kann. Solche Transformationen gehören zur Gruppe der affinen Transformationen, zu welcher jede Art von Transformation gehört, die sowohl die Abstandsverhältnisse als auch die Ko-Linearität betrifft.<sup>34</sup>

In der folgenden Abbildung 10 werden diese Probleme anhand eines Beispiels dargestellt:

---

<sup>32</sup> Vgl. Sablatnig/Zambanini/Licandro (2014), Online-Quelle [21.05.2022].

<sup>33</sup> Vgl. Gollapudi (2019), S. 27.

<sup>34</sup> Vgl. Object Recognition (2004), S. 8.

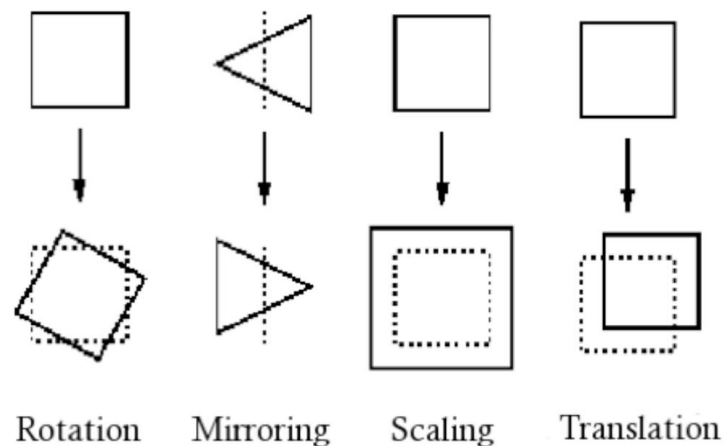


Abbildung 10: Beispiel der affinen Transformation von Objekten, Quelle: Object Recognition (2004), S. 8 (leicht modifiziert).

#### **Drehung (Rotation):**

Objekte, die unter verschiedenen Blickwinkeln erscheinen, haben sehr unterschiedliche Darstellungen, was ihre Pixel anbelangt. Ohne dies in irgendeiner Weise zu berücksichtigen, wird ein neuronales Netz identischer Muster je nach Drehung fälschlicherweise als unterschiedliche Muster erkannt.

#### **Gespiegelt (Mirroring):**

Objekte, die gespiegelt dargestellt werden, werden ähnlich wie bei der Rotation als unterschiedliche Objekte erkannt.

#### **Skalierung (Scaling):**

Unterschiedliche Größen eines Objektes können darauf zurückzuführen sein, dass Betrachtungspunkte unterschiedliche Entfernungen zu einem Vergleichsbild aufweisen. Diese Problemstellung ist für einen Computer alles andere als trivial.

#### **Verschiebung (Translation):**

Das Problem der Verschiebung ist darauf zurückzuführen, dass ein Objekt nicht immer an der gleichen Position im Bild dargestellt wird.

### **3.2.2 Lichteinfluss**

Ein immer wiederkehrendes Problem bei der Arbeit mit Bildern sind die unterschiedlichen Lichtverhältnisse, unter denen ein Bild aufgenommen wird. Der Einfluss der Beleuchtung auf die Farben ist groß und gilt insbesondere bei der farbbasierten Erkennung als eine der schwierigsten Herausforderungen. Das menschliche Auge passt sich in einem großen Bereich automatisch an die Helligkeit an. Ein Computer hat diese Fähigkeit nicht. Dies schränkt seine Fähigkeit, Farben zu erkennen, ein. Ein gutes Beispiel dafür sind die sehr beliebten Fußballroboter, die Aibos. In den meisten Fällen sind sie so programmiert, ihren Standort auf dem Spielfeld zu finden, indem sie die Farben der Objekte um sie herum erkennen. Jedes neue Spiel, das diese Roboter bestreiten, benötigt eine umfangreiche Neukalibrierung der Lichtverhältnisse. Dies



kostet wertvolle Zeit und kann mehrere Stunden in Anspruch nehmen. Jedoch bildet die zeitliche Komponente oft den Grad der Qualität. Je kürzer die Kalibration desto schlechter die Qualität. Da in dieser Arbeit mit Formen anstelle von Farben gearbeitet wird, sind die wechselnden Farben ein eher geringes Problem, jedoch bedeutet dies nicht, dass die Beleuchtung keines darstellt. Mechanismen wie die Kantenextraktion sind empfindlich gegenüber Beleuchtungsbedingungen. Kanten neigen dazu zu verschwinden, wenn die Beleuchtung schwächer wird, und der Helligkeitsunterschied von Pixeln nimmt tendenziell ab. Hellere Bilder können dazu führen, dass zu viele Kanteninformationen extrahiert werden, da bei einem helleren Bild kleine Farbveränderungen besser sichtbar werden.<sup>35</sup>

### 3.2.3 Mehrere Objekte

Das schwierigste Problem bei der Objekterkennung in realen Bildern ist, dass Objekte wahrscheinlich nicht allein auftreten, was die Objekterkennung sehr erschwert. Die Trennung von Objekten in einem Bild ist keine einfache Aufgabe. Noch schwieriger wird es, wenn sich zwei Objekte berühren oder überlappen, sodass sie möglicherweise als ein Objekt identifiziert werden. Ein solches Problem der Trennung von Objekten und des Clusters verschiedener Elemente, rechtfertigt an sich schon ein ganzes Forschungsprojekt.<sup>36</sup>

## 3.3 Darstellung eines Bildes

In dem folgenden Unterkapitel wird auf die unterschiedlichen Darstellungsmöglichkeiten eines Bildes eingegangen.

### 3.3.1 RGB-Modell

Ein großer Teil der Technologie zur Erzeugung und Darstellung von Farben basiert auf der empirischen

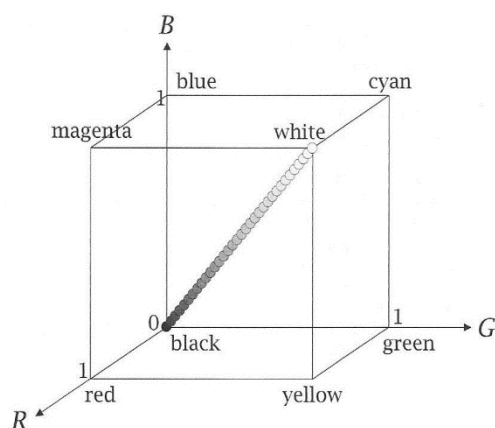


Abbildung 11: RGB-Farbwürfel, Quelle: Efford (2000), S. 28.

Beobachtung, dass eine große Vielfalt von Farben durch Mischen von rotem, grünem und blauem Licht in unterschiedlichen Verhältnissen erzielt werden kann. Aus diesem Grund werden rot (R), grün (G) und blau (B) als die Primärfarben des additiven Farbsystems bezeichnet. Nicht alle Farben können auf diese Weise erzielt werden, aber die Technik ist dennoch sehr zielführend. Der Helligkeitsgrad der drei Komponenten (RGB) in einem Farbbild, das dem RGB-Modell entspricht, ist der Wert von  $f(x,y)$  ein Vektor mit drei Komponenten, die R, G und B beschreiben. In einem normalisierten Modell variieren diese Komponenten jeweils zwischen 0,0 und 1,0.

<sup>35</sup> Vgl. Shape detection (2010), S. 3.

<sup>36</sup> Vgl. Object Recognition (2004), S. 4.

R, G und B können als orthogonale Achsen betrachtet werden, die einen dreidimensionalen Farbraum definieren. Die Grundfarben Rot, Grün und Blau sind die Ecken (1, 0, 0), (0, 1, 0) und (0, 0, 1) und die Farben Cyan, Magenta und Gelb sind die gegenüberliegenden Ecken. Schwarz befindet sich im Ursprung und Weiß in der Ecke, die am weitesten vom Ursprung entfernt ist. Da jede der drei Komponenten normalerweise mit 8 Bit quantisiert wird, wird ein Bild, das aus diesen Komponenten besteht, üblicherweise als 24-Bit-Farbe bezeichnet.<sup>37</sup> Die Abbildung 11 zeigt diesen RGB-Farbwürfel.

### 3.3.2 Graustufenbild

Ein Bild ist eine visuelle Darstellung eines Objekts, welches die Form eines zweidimensionalen Signals aufweist, das durch  $f(x, y)$  definiert ist, wobei  $f$  die Intensität darstellt. Die Farbinformationen werden durch das RGB-Format dargestellt.<sup>38</sup> In einem RGB-Bild können alle diese Informationen eines Bildes gespeichert werden, da jedoch in dieser Arbeit auch auf die Kantenextraktion eingegangen wird, kann dies nicht praktikabel sein. Dadurch ist es von Vorteil, ein Farbbild in ein Graustufenbild (GSB) umzuwandeln.<sup>39</sup> Zusätzlich wird auch nur die Diagonale zwischen weiß und schwarz des Farbwürfels, wie in Abbildung 11 dargestellt, verwendet. Dadurch ergibt sich eine Speicherplatzreduktion pro Bild von 24 Bit auf 8 Bit. Es gibt drei einfache mathematische Möglichkeiten ein Farbbild in ein GSB umzuwandeln:

1. Einfacher Mittelwert:

$$Gw(x) = \frac{x.R + x.G + x.B}{3} \quad (3.1)$$

$Gw(x)$	Grauwert
$x.R$	Rotwert
$x.G$	Grünwert
$x.B$	Blauwert

2. Mittelwert aus Maximum und Minimum

$$Gw(x) = \frac{\min(x.R, x.G, x.B) + \max(x.R, x.G, x.B)}{2} \quad (3.2)$$

3. Gewichteter Mittelwert

$$Gw(x) = Gw(x) = 0.3 * x.R + 0.59 * x.G + 0.11 * x.B \quad (3.3)$$

Der gewichtete Mittelwert kann variieren. Die oben genannten Werte sind Beispiele, die aus dem Bildbearbeitungstool Gimp standardmäßig eingestellt sind.<sup>40</sup>

---

<sup>37</sup> Vgl. Efford (2000), S. 27f.

<sup>38</sup> Vgl. Grayscale Image (2010), S. 1.

<sup>39</sup> Vgl. Hermes (2005), S. 29.

<sup>40</sup> Vgl. Gimp (2010), Online-Quelle [21.05.2022].

### 3.3.3 Das Binärbild

Aufgrund ihres Pixelwertebereichs werden Bilder in Farb-, Graustufen- und Binärbilder unterteilt. Bei einem Binärbild (BB) weisen die Wertebereiche der Pixel nur 0 und 1 (logisch 0 und logisch 1) auf. Dies bedeutet, dass ein BB nur aus Schwarz- und Weißanteilen besteht. Binärbilder werden meistens für die

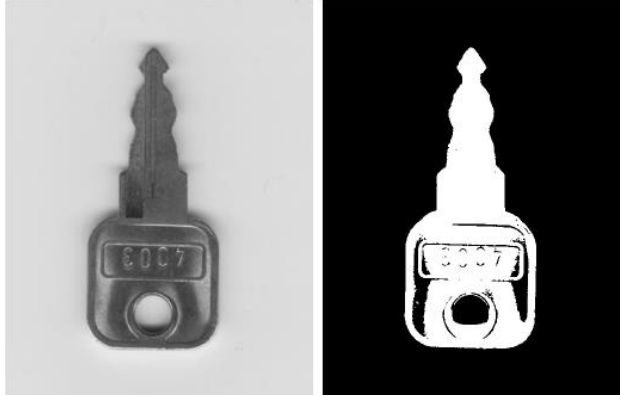


Abbildung 12: Binärbild aus einem Grauwertbild, Quelle: Hermes (2005), S. 30.

Vorverarbeitung (z.B. Rauschminderung) oder Segmentierungen eingesetzt. Für die Erzeugung eines BB werden meist GSB segmentiert. Die häufigste und einfachste Methode ist dabei das sogenannte Schwellenwertverfahren (Thresholding). Hier werden Intensitätswerte über einer gewissen Schwelle des Grauwerts, welche variabel definiert ist, auf 1 gesetzt und alle darunter liegenden Werte auf 0. Man spricht hier von „Binarisierung“.<sup>41</sup> Die Abbildung 12 zeigt eine GSB eines Schlüssels, welches in ein BB umgewandelt wurde.

## 3.4 Methoden zur Bildverarbeitung

In dem vorigen Kapitel 3.2 wurde auf die Herausforderungen der Bildverarbeitung und Objekterkennung hingewiesen. In diesem Kapitel wird auf Methoden, Funktionen und Filter eingegangen, um die Probleme der CV zu minimieren.

Es gibt viele Arten und Methoden, Objekte zu erkennen, welche von einfachen Funktionen bis hin zu Machine-Learning gehen. Um eine Übersicht zu geben, wird nur auf Methoden und Verarbeitungsschritte eingegangen, die eine effiziente Erkennung von SD ermöglichen. Um ein gemeinsames Verständnis zu haben, werden auch grundlegende Informationen bereitgestellt.

### 3.4.1 Filter

Der Unterschied zwischen Filter und z.B. Binarisierung liegt darin, dass nicht nur einzelne Pixelwerte verändert werden, sondern ganze Pixelgruppen. Dies bedeutet, es werden Pixel und deren angrenzenden Pixel herangezogen und mathematisch verändert.<sup>42</sup> In den folgenden Unterkapitel wird auf grundlegende Filtermethoden eingegangen.

<sup>41</sup> Vgl. Hermes (2005), S. 22.

<sup>42</sup> Vgl. Erhardt (2008), S. 154.

### 3.4.1.1 Glättung

Zunächst soll die Aufgabe der Bildglättung oder des Weichzeichnens (engl. „Blurring“) genauer dargestellt werden. Bilder sehen vor allem dort scharf aus, wo die lokale Intensität stark ansteigt oder abfällt (d.h. dort, wo der Unterschied zwischen benachbarten Pixeln groß ist). Andererseits empfinden wir ein Bild als unscharf oder verschwommen, wenn die lokale Intensität sehr glatt ist. Eine erste Idee zur Glättung eines Bildes könnte also darin bestehen, einfach jeden Pixel durch den Durchschnitt seiner benachbarten Pixel zu ersetzen.<sup>43</sup>



Abbildung 13: Blurring eines Bildes, Quelle: Burger/Burge (2009), S. 98.

Die Abbildung 13 zeigt eine Glättung eines Bildes. Anfänglich sieht dieses Bild sehr verschwommen aus, jedoch ist diese Methode sehr nützlich, um wichtige Informationen hervorzuheben. Starke Kontraste werden erhalten und weiche Übergänge verschwinden eher ineinander. So kann man den Bus, wie in Abbildung 13 ersichtlich, von seinem Hintergrund abheben. Zur Bestimmung des neuen Pixelwertes im geglätteten Bild  $I(u, v)$  verwenden wir den ursprünglichen Pixel  $I(u, v) = p_0$  an der gleichen Position sowie seine acht Nachbarpixel  $p_1, p_2, \dots, p_8$ , um das arithmetische Mittel dieser neun Werte zu bilden.<sup>44</sup>

$$I'(u, v) = \frac{p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 + p_9}{9} \quad (3.4) \quad \begin{array}{ll} I'(u, v) & \text{Neuer Pixel} \\ p(x) & \text{Pixel} \end{array}$$

### 3.4.1.2 Die Filtermatrix

Für jeden linearen Filter wird die Größe und die Form der Stützregion sowie die Gewichte der einzelnen Pixel durch die Filtermatrix oder Filtermaske  $H(i, j)$  festgelegt. Die Größe der Matrix  $H$  entspricht der Größe der Filterregion und jedes Element  $H(i, j)$  gibt das Gewicht des entsprechenden Pixels in der Summierung an. Dadurch entsteht ein eindeutiger Mittelpunkt der Matrix der auch Hot Spot genannt wird.<sup>45</sup>

Für den oben genannten Blurring-Filter sieht die Filtermatrix wie folgt aus:

<sup>43</sup> Vgl. Burger/Burge (2009), S. 97f.

<sup>44</sup> Vgl. Burger/Burge (2009), S. 98.

<sup>45</sup> Vgl. Burger/Burge (2009), S. 100.

$$H(i, j) = \begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix} \quad (3.5)$$

$i$  und  $j$  stehen in diesem Fall für die Indizes der Koordinaten, von wo der Hot Spot ausgeht. Die Abbildung 14 soll dies verdeutlichen:

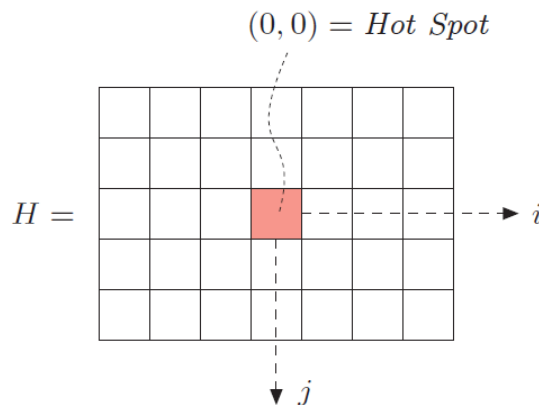


Abbildung 14: Filtermatrix und Hot Spot, Quelle: Burger/Burge (2009). S. 100.

### 3.4.1.3 Anwendung eines Filters

Bei einem Filter wird das Ergebnis eindeutig und vollständig durch die Koeffizienten der Filtermatrix spezifiziert. Um einen Filter anzuwenden, bedarf es drei wesentlicher Schritte.<sup>46</sup> Diese Schritte werden auch in Abbildung 15 dargestellt.

1. Die Filtermatrix  $H$  wird über das Originalbild  $I$  so verschoben, dass ihr Ursprung  $H(0,0)$  mit der aktuellen Bildposition  $(u, v)$  zusammenfällt.
2. Alle Filterkoeffizienten  $H(i, j)$  werden mit dem entsprechenden Bildelement  $I(u + i, v + i)$  multipliziert und die Ergebnisse werden addiert.
3. Schließlich wird die resultierende Summe an der aktuellen Position im neuen Bild gespeichert  $I'(u, v)$ .

<sup>46</sup> Vgl. Burger/Burge (2009). S. 101.

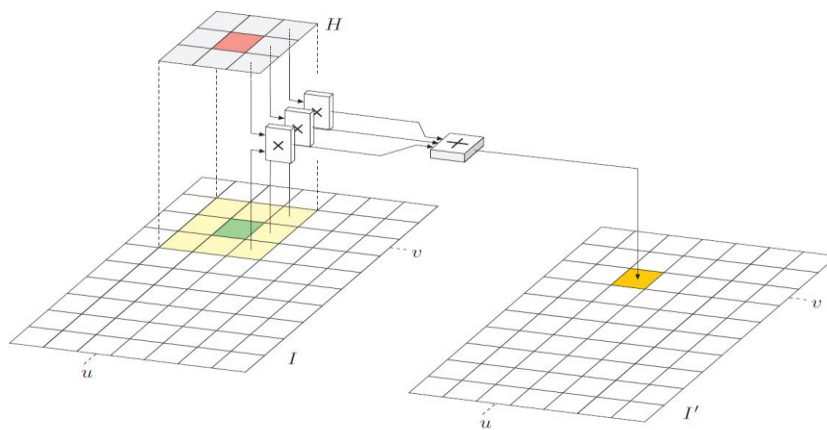


Abbildung 15: Anwendung einer Filter-Matrix, Quelle: Burger/Burge (2009), S. 101.

### 3.4.1.4 Grenzen von Filter

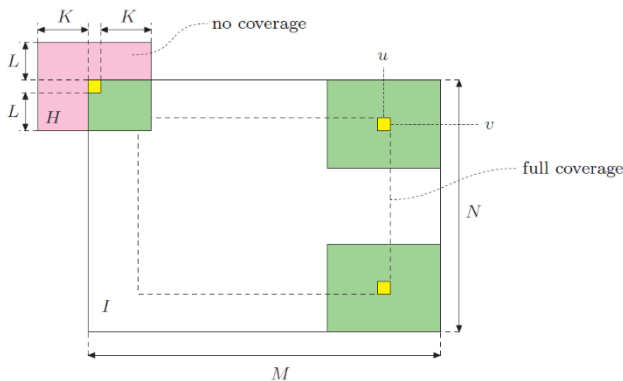


Abbildung 16: Grenzen von Filtern, Quelle: Burger/Burge (2009), S. 107.

Wie im vorigen Kapitel angesprochen, werden Filter so eingesetzt, dass sie den Hot Spot und deren umliegenden Pixel zur Errechnung des neuen Pixels heranziehen. Bei genauerer Betrachtung kann dies zu Problemen am Bildrand führen, wie in Abbildung 16 dargestellt. Dies lässt sich umgehen, indem der Filter automatisch die Werte der umliegenden Pixel den fehlenden Pixel zuweist.<sup>47</sup>

## 3.4.2 Kantendetektion

Kanten und Konturen spielen beim menschlichen Sehen und wahrscheinlich auch bei vielen anderen biologischen Sehsystemen eine herausragende Rolle. Kanten sind nicht nur visuell auffällig, sondern es ist oft möglich, eine komplette Figur aus wenigen Schlüssellinien zu beschreiben oder zu rekonstruieren. Kanten lassen sich grob als Bildpositionen beschreiben, an denen sich die lokale Intensität entlang einer bestimmten Orientierung deutlich ändert. Je stärker die lokale Intensitätsänderung ist, desto höher ist die Wahrscheinlichkeit, dass an dieser Stelle eine Kante vorliegt.<sup>48</sup> In dem folgenden Kapitel wird auf den Grundprinzipien der Kantendetektion eingegangen, welche es ermöglichen sollen, einen DP anhand seiner Kontur zu erkennen bzw. diesen zu lokalisieren.

<sup>47</sup> Vgl. Burger/Burge (2009), S. 106f.

<sup>48</sup> Vgl. Burger/Burge (2009), S. 133.

Die ideale Kante ist eine sprunghafte maximale Änderung in einem GSB oder BB. So wird bei einem GSB der Faktor um ein Vielfaches oder bei einem Binärbild um das logisch Invertierte verändert. Bei einem GSB kann der Veränderungsfaktor natürlich variieren und so bedarf es eines variabel definierten Faktors, um eine Kante zu erkennen. In dem folgenden Bild soll eine ideale Kante eines GSB dargestellt werden:<sup>49</sup>



Abbildung 17: Ideale Kante anhand eines Grauwertes, Quelle: Hermes (2005) (leicht modifiziert), S. 70.

Anhand eines Ausschnittes  $P$  eines GSB und deren Filter Matrix, würde eine ideale Kante wie folgt aussehen.

$$P = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & 50 & 50 & 50 & 150 & 150 & 150 & \dots \\ \dots & 50 & \mathbf{50} & 50 & 150 & 150 & 150 & \dots \\ \dots & 50 & 50 & 50 & 150 & 150 & 150 & \dots \\ \dots & 50 & 50 & 50 & 150 & 150 & 150 & \dots \\ \dots & 50 & 50 & 50 & 150 & 150 & 150 & \dots \\ \dots & 50 & 50 & 50 & 150 & 150 & 150 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \quad (3.6)$$

Um hier die Kante hervorzuheben, ist es möglich mit einem Laplace Operator  $L$  zu arbeiten.

$$L = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad (3.7)$$

An den fett markierten Hot Spots in Formel 3.6 kommt dieser zum Einsatz.

$$P = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & 50_0 & 50_{-1} & 50_0 & 150 & 150 & 150 & \dots \\ \dots & 50_{-1} & \mathbf{50_{-4}} & 50_{-1} & 150 & 150 & 150 & \dots \\ \dots & 50_0 & 50_{-1} & 50_0 & 150 & 150 & 150 & \dots \\ \dots & 50 & 50 & 50 & 150 & 150 & 150 & \dots \\ \dots & 50 & 50_0 & 50_{-1} & 150_0 & 150 & 150 & \dots \\ \dots & 50 & 50_{-1} & \mathbf{50_{-4}} & 50_{-1} & 150 & 150 & \dots \\ \dots & 50 & 50_0 & 50_{-1} & 150_0 & 150 & 150 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \quad (3.8)$$

Wendet man nun den Laplace Operator an so erhält man folgende Matrix:

<sup>49</sup> Vgl. Hermes (2005), S. 57.

$$P = \begin{pmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & 0 & 0 & 100 & 100 & 0 & 0 & \dots \\ \dots & 0 & 0 & 100 & 100 & 0 & 0 & \dots \\ \dots & 0 & 0 & 100 & 100 & 0 & 0 & \dots \\ \dots & 0 & 0 & 100 & 100 & 0 & 0 & \dots \\ \dots & 0 & 0 & 100 & 100 & 0 & 0 & \dots \\ \dots & 0 & 0 & 100 & 100 & 0 & 0 & \dots \\ \dots & 0 & 0 & 100 & 100 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \quad (3.9)$$

Der Hotspot, wie oben dargestellt, wird auf 0 gesetzt. Die umliegenden Werte hingegen werden auf -100 gesetzt. Da es im GSB keine negativen Zahlen gibt, wird in diesem Fall der Betrag herangezogen. Dies ist äußerst wichtig bei der Kantenextraktion, da dadurch eine senkrecht verlaufende Kante entsteht.<sup>50</sup>

---

<sup>50</sup> Vgl. Hermes (2005), S. 60.



## 4 OPENCV

In dem folgenden Kapitel wird auf die Bildverarbeitungsbibliothek OpenCV (Open Source Computer Vision) (OCV) eingegangen.

### 4.1 Einführung in OCV

OCV ist eine Open Source-Bildverarbeitungsbibliothek. Die Bibliothek wurde in C und C++ entwickelt und läuft unter Linux, Windows und Mac OS X. Es gibt Schnittstellen zu Python, Ruby, Matlab sowie andere höhere Programmiersprachen. OCV wurde entwickelt, um die Effizienz des Rechners auszunutzen und Echtzeitanwendungen abzuwickeln. Da OCV in C/C++ entwickelt worden ist, kann diese die Vorteile von Mehrkernarchitekturen nutzen. Dadurch wird die Performance gesteigert und Prozesse optimiert. Der Grundgedanke von OCV ist es, eine einfach zu bedienende „Computer-Vision (CV)“-Infrastruktur zur Verfügung zu stellen, um schnelle Bildverarbeitungsalgorithmen zu erstellen.<sup>51</sup> Da die Bibliothek frei zu Verfügung gestellt wird, gibt es eine große Anzahl von Entwicklungen, die leicht zugänglich sind.

Zurzeit verfügt die Bibliothek über mehr als 2500 optimierte Algorithmen. Zusätzlich gibt es eine große Auswahl von klassischen und modernen Algorithmen, welche auf CV und „Machine-Learning“ abzielen. Diese dienen zur Erkennung von Gesichtern, Identifizierung von Objekten und Klassifizierung von menschlichen Interaktionen in Videos. Weitere Anwendungen sind Extraktionen von 3D-Modellen aus Objekten, Zusammenführung oder Nachbearbeitung von Bildern. Die große Auswahl von Filtern ermöglicht dem Anwender, Bilder zu verbessern oder nachzubearbeiten. Schätzungen zufolge gibt es allein in Deutschland über 47.000 OCV-Benutzer und mehr als 18 Millionen Downloads der Bibliothek.<sup>52</sup>

### 4.2 Stakeholders von OCV

Stakeholder sind die Personen, Teams oder Organisationen, die ein Interesse an der Realisierung eines Softwaresystems haben.<sup>53</sup> Von den 11 Arten von Stakeholdern, die in dem Buch von Rozanski und Woods vorgestellt werden, sind 5 auf OCV anwendbar. In der folgenden Tabelle werden die 5 Stakeholder dargestellt und die jeweiligen wichtigsten beschrieben:

---

<sup>51</sup> Vgl. Bradski/Kaebler (2008), S. 1.

<sup>52</sup> Vgl. Team (2022), Online-Quelle [21.05.2022].

<sup>53</sup> Vgl. Rozanski/Woods (2012). S. 24.

Stakeholder	Beschreibung
<b>Kommunikator</b>	Maksim Shabunin ist der Hauptkommunikator bei OCv. Er hat systematisch an der Dokumentation gearbeitet, wobei diese Tätigkeit sonst von der gesamten Gemeinschaft gepflegt wird. <sup>54</sup>
<b>Entwickler</b>	Die Entwickler Alexander Alekhin, Maksim Shabunin und Steven Puttemans sind die Hauptentwickler von OCV, wobei insgesamt mehr als 500 Benutzer zu dem Projekt beigetragen haben. Die meisten Entwickler haben dazu einen Beitrag geleistet, weil sie bei der Verwendung der Bibliothek auf ein Problem gestoßen sind und eine Lösung gefunden haben, um dieses zu beheben. All diese Änderungen werden in den nächsten Versionen von OCV berücksichtigt. <sup>55</sup>
<b>Tester</b>	Tester werden nicht explizit eingesetzt, stattdessen schreiben die Entwickler die Tests selbst, wenn sie neue Funktionen implementieren. Zusätzlich werden Pushbots zur Fehlerabsicherung eingesetzt.
<b>Benutzer</b>	Die Benutzer setzen sich aus kleinen und großen Unternehmen, Regierungsbehörden und Software-Ingenieuren zusammen. Einige Beispiele für Unternehmen, die OpenCV nutzen, sind Google, Intel, IBM und Toyota, wobei die Gesamtzahl der Downloads auf Firmenbases auf mehr als 7 Millionen geschätzt wird. <sup>56</sup>
<b>Integrator</b>	Alexander Alekhin, Maksim Shabunin und Steven Puttemans sind die Integratoren. Sie versuchen, das Projekt systematisch zu entwickeln und dabei die Codequalität, den Codestil und die Architektur beizubehalten. <sup>57</sup>

Tabelle 5: Stakeholder von OCV, Quelle: Eigene Darstellung.

### 4.3 Kontextübersicht von OCV

Die Kontextübersicht von OCV definiert die Beziehungen, Interaktionen und Abhängigkeiten zwischen OCV und seiner Umgebung. Diese gibt einen umfassenden Überblick über das gesamte System und ist daher für alle Beteiligten nützlich. Dieser Überblick wird in Abbildung 18 dargestellt.

<sup>54</sup> Vgl. OpenCV (2022), Online-Quelle [21.05.2022].

<sup>55</sup> Vgl. OpenCV (2022), Online-Quelle [21.05.2022].

<sup>56</sup> Vgl. OpenCV (2022), Online-Quelle [21.05.2022].

<sup>57</sup> Vgl. Bjarki/Shruthi/Renukaprasad (2022), Online-Quelle [21.05.2022].

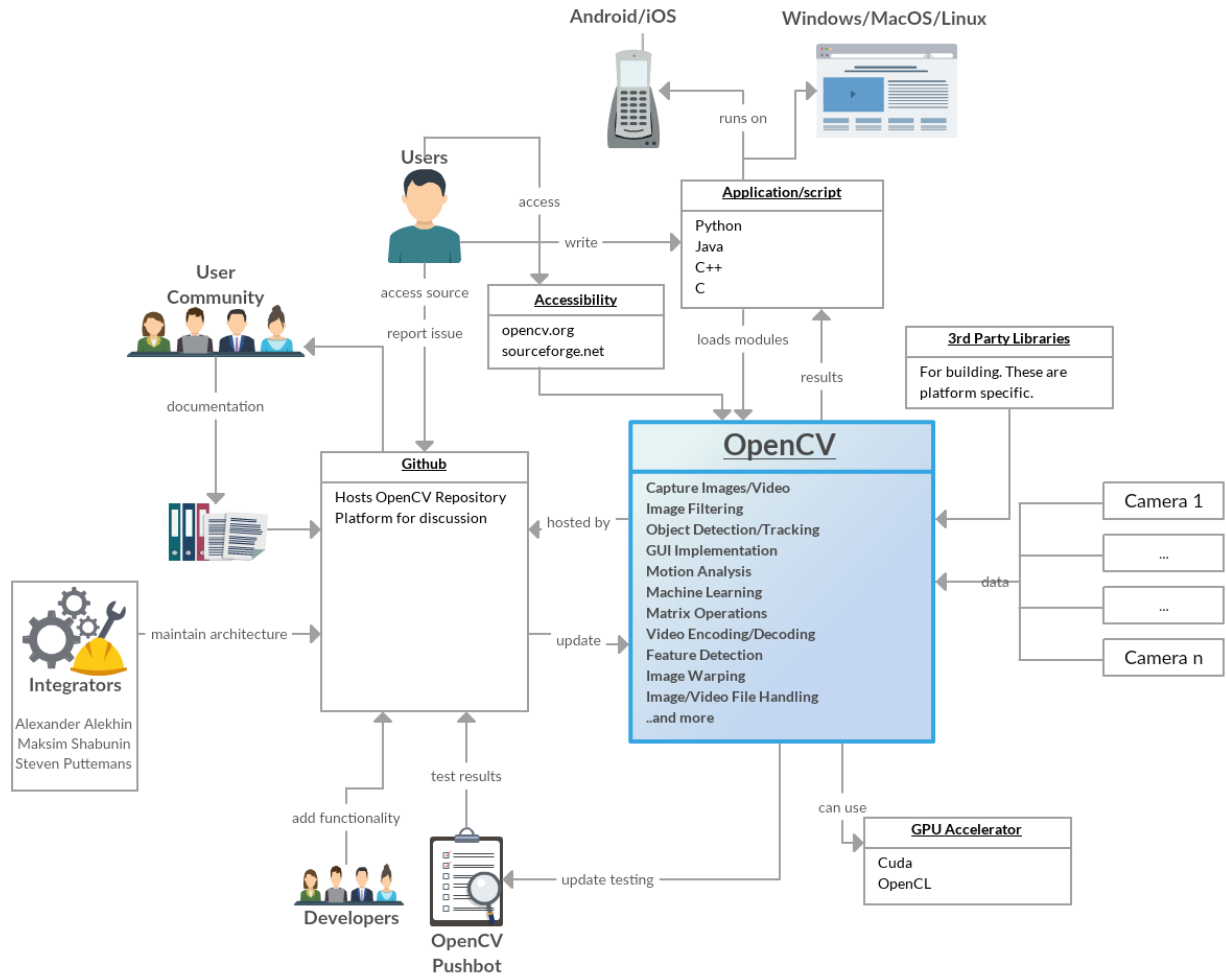


Abbildung 18: Übersicht OPC, Quelle: Bjarki/Shruthi/Renukprasad (2022), Online-Quelle [21.05.2022].

OCV ist eng mit GitHub verknüpft, in welcher der Quellcode gehostet wird. Benutzer können diesen klonen oder herunterladen. Diskussionen über Problemlösungsansätze und fehlende oder fehlerhafte Funktionen finden ebenfalls über GitHub statt. Entwickler können zum Projekt OCV beitragen, indem sie Anfragen stellen und Verbesserungen vorschlagen, die dann von den Integratoren bearbeitet werden. Die Dokumentation wird von der Benutzergemeinschaft über GitHub zur Verfügung gestellt, wobei Maksim Shabunin der Hauptverantwortliche in diesem Bereich ist. Alle im Abschnitt "Stakeholder" besprochenen Interessengruppen sind auch im Modell der Kontextübersicht enthalten und alle von ihnen nutzen GitHub als Mittel zum Zugriff auf Informationen über OCV.<sup>58</sup>

OCV ist eine hochgradig optimierte Echtzeitbibliothek und muss daher zahlreiche Hardwarebeschleuniger und externe Geräte unterstützen. Sie unterstützt CUDA, eine Plattform für parallele Berechnungen auf NVIDIAS GPUs. Über 375 Millionen CUDA-fähige GPUs sind in modernen Computern vorhanden, die es den Entwicklern ermöglichen, ihre Algorithmen auf GPUs auszuführen. OCV unterstützt auch andere

<sup>58</sup> Vgl. Bjarki/Shruthi/Renukprasad (2022), Online-Quelle [21.05.2022].

parallele Programmierbibliothek, nämlich OpenCL (Open Computing Language), einen quelloffenen, lizenzgebührenfreien Standard für parallele Programmierung der Khronos-Group. Zusätzlich ist OCV eine plattformübergreifende, mehrsprachige Bibliothek, die alle wichtigen Betriebssysteme sowie die Programmiersprachen C++, Java und Python unterstützt und daher auf zahlreiche Bibliotheken von Drittanbietern zurückgreifen kann. Sie unterstützt bis auf wenige Ausnahmen alle Kameratypen. Diese sind die wichtigsten Datenquellen für Bildverarbeitungsanwendungen.<sup>59</sup>

## 4.4 Programmiersprachen für OCV

In diesem Unterkapitel wird eine Übersicht über die zu verwendeten Programmiersprachen gegeben.

Die am häufigsten verwendete Sprache in OCV ist C++.<sup>60</sup> Es gibt jedoch eine beträchtliche Menge an CUDA- OpenCL- sowie Java- und Python-Codes, die zur Verbesserung der Leistung oder Portabilität benötigt werden. Außerdem werden viele andere Sprachen nur für sehr spezifische Aufgaben verwendet.<sup>61</sup>

In der folgenden Abbildung 19 wird die Verteilung der benutzten Programmiersprachen für OCV dargestellt:

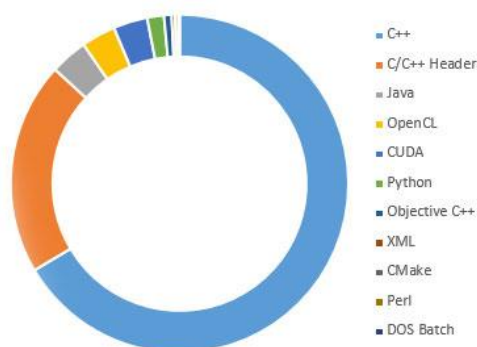


Abbildung 19: Übersicht der benutzten Programmiersprachen für OCV, Quelle: Bjarki/Shruthi/Renukaprasad (2022), Online-Quelle [21.05.2022].

## 4.5 OCV Strukturübersicht

In diesem Unterkapitel wird auf die allgemeine Aufbaustruktur von OCV eingegangen.

OCV ist grob in fünf Hauptkomponenten unterteilt, von denen vier in Abbildung 20 dargestellt sind. Die CV-Komponente enthält die grundlegenden Bildverarbeitungs- und übergeordneten Bildverarbeitungsalgorithmen. MLL (Machine Learning Library) ist eine Bibliothek und beinhaltet viele statistische Klassifizierungs- und Clustering-Werkzeuge. HighGUI enthält Ein- und Ausgaberroutinen sowie Funktionen zum Speichern und Laden von Videos und Bildern. CXCore enthält die grundlegenden Datenstrukturen und Inhalte.<sup>62</sup>

<sup>59</sup> Vgl. Bjarki/Shruthi/Renukaprasad (2022), Online-Quelle [21.05.2022].

<sup>60</sup> Vgl. Qingcang/Chenga/Chengb/Xiaodong (2004), S. 1.

<sup>61</sup> Vgl. Bjarki/Shruthi/Renukaprasad (2022), Online-Quelle [21.05.2022].

<sup>62</sup> Vgl. Bradski/Kaebler (2008), S. 13.

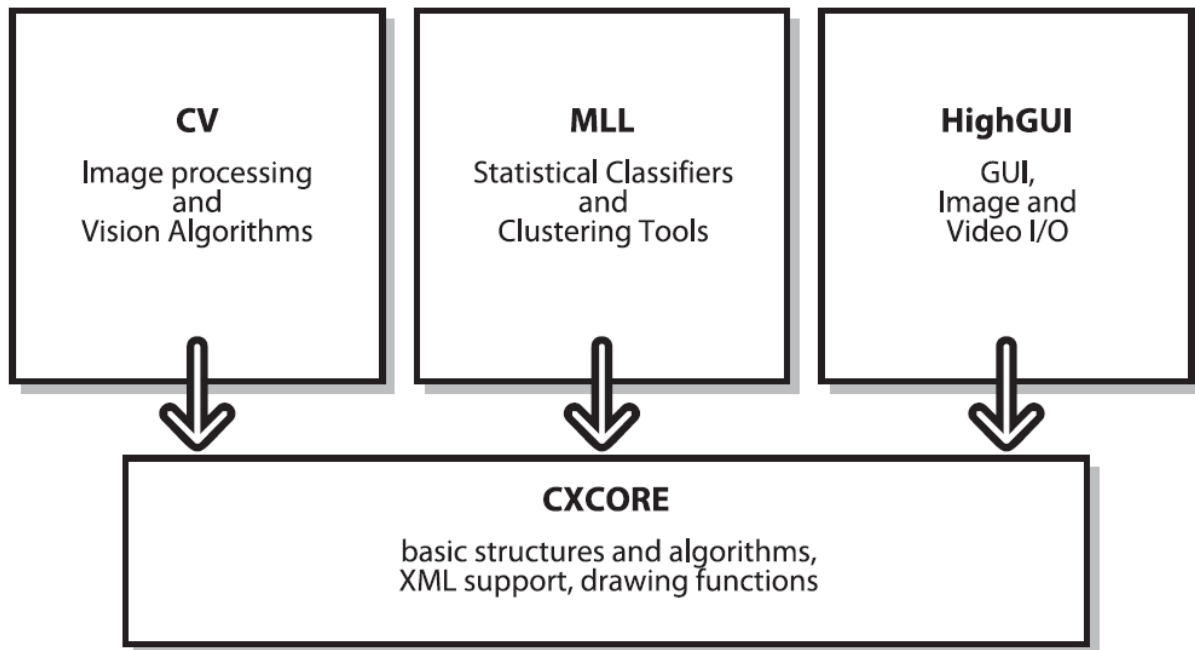


Abbildung 20: Basisstruktur von OCV, Quelle: Bradski/Kaebler (2008), S. 13.

In Abbildung 20 ist CvAux nicht enthalten, welches experimentelle Algorithmen (Hintergrund-Vordergrundsegmentierung) enthält. CvAux ist nicht gut dokumentiert, deckt jedoch folgende Anwendungen und Funktionen ab:<sup>63</sup>

- Rechenintensive Erkennungsverfahren und Template-Matching
- Hidden-Markov-Modelle für statistische Auswertungen
- Gestenerkennung durch Unterstützung der Stereovision
- Erweiterungen für Delaunay-Triangulation
- Stereovision
- Formvergleich mit definierten Konturen
- Augen- und Mundverfolgung
- 3D-Verfolgung
- Auffinden von Skeletten (zentralen Linien) von Objekten in einer Szene
- Verzerrung von Zwischenansichten zwischen zwei Kameraansichten
- Hintergrund-Vordergrund-Segmentierung
- Kamerakalibrierung

## 4.6 Emgu CV

Emgu CV (ECV) ist ein plattformübergreifender .NET-Wrapper für die OCV-Bildverarbeitungsbibliothek. Damit können OCV-Funktionen von .NET-kompatiblen Sprachen aus aufgerufen werden. Der Wrapper

<sup>63</sup> Vgl. Bradski/Kaebler (2008), S. 16.

kann mit Visual Studio und Unity kompiliert werden und läuft auf Windows, Linux, Mac OS, iOS und Android. Es ist viel Aufwand betrieben worden, um eine reine C#-Implementierung umzusetzen, da Header importiert werden müssen, verglichen mit einer C++-Implementierung, bei der Header-Dateien einfach eingebunden werden können.<sup>64</sup>

---

<sup>64</sup> Vgl. MediaWiki (2022), Online-Quelle [21.05.2022].

## 5 ANFORDERUNGSANALYSE UND KONZEPTFINDUNG

Ziel dieser Arbeit ist es, ein System zu entwickeln, welches geworfene DP auf einer DS erkennt und mittels Positionsbestimmung dem getroffenen Segment und deren Punkte zuweist. Hierfür sollen kontinuierlich Bilder von Kameras eingelesen werden, die bei Erkennung einer Veränderung einen Auswertalgorithmus starten sollen. Das zu suchende Objekt, der DP, soll mittels Filterung und Kantenextraktion bestimmt werden. Das Objekt kann durch seinen rechteckigen bis leicht trapezförmigen Schaft klassifiziert werden. Da sich bis zu drei DP gleichzeitig auf der DS befinden können, muss ein System konzipiert werden, welches die DS aus mehreren Blickwinkel betrachtet, um die Überdeckung minimal zu halten. Zusätzlich fließen Erkenntnisse aus der Bachelorarbeit „Optische Auswertung von Steeldartpfeilen auf einer Dartscheibe“ ein, welche den Proof-of-Concept dieser Arbeit darstellt.

### 5.1 Grundidee

Die Grundidee des Systems zielt darauf ab, mittels eines Vierkamerasystems ein redundantes System zu entwickeln, das wenig beeinflussbar von Lichtverhältnissen ist. Dazu soll eine Hauptkamera die Veränderungen auf einer DS detektieren. Durch Erzeugen eines GSB und BB können äußerliche Einflüsse minimiert werden. Je nach Position und Überlappung der DP soll ein Priorisierungsalgorithmus die je zwei Kameras, welche normal aufeinander stehen, aktivieren und den Auswertalgorithmus starten. Um diese Funktionen zu testen, soll eine Testumgebung in C# entwickelt werden, welche mithilfe OCV-Bibliothek die einzelnen Funktionen und Filtereinstellungen herauskristallisiert. Auf Basis dieser Erkenntnisse kann dann der Auswertalgorithmus erarbeitet werden. Nach erfolgreicher Umsetzung auf einem Standrechner und einer Windows Umgebung, soll das Programm auf einen EPC übertragen werden. Dieser soll mittels Remoteverbindung die Programmoberfläche übertragen.

Um dieses Konzept umzusetzen, soll eine Aufbautestumgebung für die Kameras erarbeitet und produziert werden. Ziel ist darüber hinaus, dass der Aufbau stabil und für unterschiedliche DS anwendbar ist.

### 5.2 Anforderungsanalyse

Um einen Überblick über die zu entwickelten Funktionen zu erlangen, muss anfänglich eine Anforderungsanalyse durchgeführt werden. Wie im Kapitel 5.1 beschrieben, soll einerseits die mechanische Testumgebung und andererseits die Softwaretestumgebung analysiert werden.

#### 5.2.1 Mechanische Anforderungsanalyse

In diesem Kapitel wird auf den praktischen Aufbau eingegangen. Um die Anforderungen besser erklären zu können, werden Abbildungen verwendet, die eine mögliche Umsetzung wiedergeben sollen.

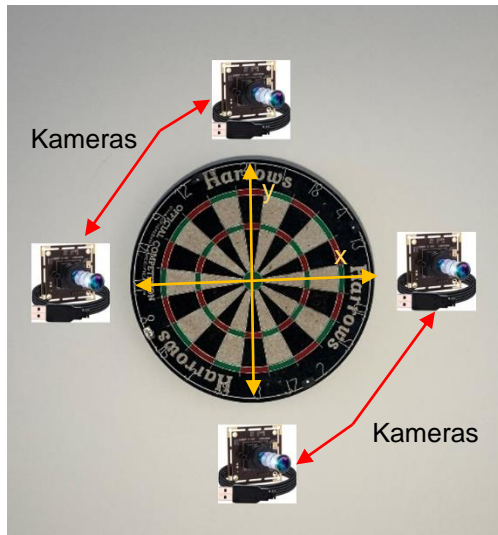


Abbildung 21: Positionierung eines Vierkamerasystems um die DS, Quelle: Eigene Darstellung.

### Rundumblick:

Um die DS von allen Blickwinkeln genau zu erfassen, sollen vier Kameras im 90° Winkel zueinander positioniert werden. Die komplette DS kann abgedeckt und genau visuell erfasst werden. Zusätzlich können auch die Regionen erfasst werden, die nicht auf der DS liegen. Die Abbildung 21 zeigt eine mögliche Positionierung der Kameras um die DS, um den Rundumblick zu ermöglichen.

### Auswertung des Koordinatensystems und Fischaugeneffekt:

Wie in Abbildung 21 dargestellt, soll ein Koordinatensystem zur Auswertung herangezogen werden. Um dies zu ermöglichen, muss ein mögliches Verhältnis zwischen dem Abstand der Kameras zur Dartscheibe und dem

Erfassungswinkel der Kameras gefunden werden. Kameraobjektive haben Einschränkungen in Bezug auf ihren Erfassungswinkels. Wenn man einen Erfassungswinkel über 110° überschreitet, kann es bei konventionellen USB-Kameras zu dem sogenannten Fischaugeneffekt kommen. Dieser Effekt wird durch das Fischaugenobjektiv herbeigeführt. Ein Fischaugenobjektiv ist eine Art Ultraweitwinkelobjektiv, das die

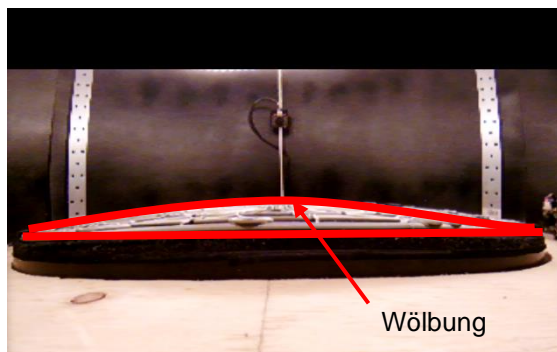


Abbildung 22: Fischaugeneffekt einer Kamera mit mehr als 110° Erfassungswinkel, Quelle: Auswertung von Steeldartpfeilen (2020), S. 34 (leicht modifiziert).

Szene oder das Motiv verzerrt, um ein halbkugelförmiges (oder breites Panorama-)Bild zu erzeugen. Es erzeugt Bilder mit einem ähnlichen Effekt, wie wenn man durch ein Guckloch in einer Tür schaut - verzerrt, kreisförmig und weitwinkelig.<sup>65</sup> Um dem Fischaugeneffekt entgegenzuwirken, müssen Kameras eingesetzt werden, die nur einen Weitwinkeleffekt haben und maximal einen Erfassungswinkel von 110° aufweisen. Diese Erkenntnisse sind im Zuge der Bachelorarbeit „Optische Auswertung von Steeldartpfeilen auf einer Dartscheibe“ erarbeitet worden.<sup>66</sup> Die Abbildung 22 zeigt genau diesen Effekt, der zufolge hat, dass keine exakte Berechnung

des Koordinatensystems erzielt werden und deswegen auch keine exakte Position des DP auf der DS festgestellt werden kann.

### Überlappung:

Eine sehr große Herausforderung in der Erkennung von DP auf einer DS ist es, drei DP separat zu erfassen. In den meisten Fällen ist die Überlappung der DP nicht gegeben, da sich die Pfeile in einem

<sup>65</sup> Vgl. Canon (2022), Online-Quelle [21.05.2022].

<sup>66</sup> Vgl. Auswertung von Steeldartpfeilen (2020), S. 34.



gewissen Abstand voneinander befinden. Jedoch kann es vorkommen, dass die Pfeile genau in dem gleichen Winkel, nah beieinander und in der gleichen Ebene auf der DS stecken (siehe Abbildung 23). Um hier entgegenzuwirken, kann das Vierkammersystem eingesetzt werden, da die zwei DP von der jeweils gegenüberliegenden Kamera, wie in Abbildung 21 dargestellt, erfasst werden können.



Abbildung 23: Überlappung von zwei Dartpfeilen, Quelle: Auswertung von Steeldartpfeilen (2020), S. 20.

### Lichteinflüsse:

Wie in Kapitel 3.2.2 beschrieben, sind die Lichteinflüsse in der Bildverarbeitung ein entscheidendes Problem. Einerseits können sich verändernde Lichtverhältnisse unvorhergesehene Probleme im Detektionsalgorithmus hervorrufen, weshalb der Algorithmus detektieren könnte, dass sich das Bild verändert hat und sich ein neuer DP auf dem DS befindet. Andererseits können Schatten von DP den Auswertalgorithmus fälschlicherweise erkennen lassen, dass sich zusätzliche Pfeile auf der Scheibe befinden. Um dieses Problem zu beheben, kann eine Lichtquelle eingesetzt werden, die die komplette DS rundum mit Licht bestrahlt. Darüber hinaus können damit auch die von den Pfeilen geworfenen Schatten minimiert werden. Zusätzlich zur Beleuchtung ist es auch möglich, die Kameras so zu positionieren, dass sie Schatten nicht mehr wahrnehmen. Dies bedeutet, dass sie die Oberfläche der DS nicht mehr aufnehmen. Wie in der Abbildung 23 im Bild links oben dargestellt, würde eine Kamera, die sich in der gleichen Ebene wie die DS befindet, einen Schatten nicht mehr aufnehmen.

### Schutzvorrichtungen:

Da es vorkommen kann, dass sich DP von der DS lösen oder vom Metallgitter auf der DS abprallen, müssen die Kameras und die Elektronik geschützt werden. Dies kann einerseits durch Positionierung und



Abbildung 24: Dartring zum Schutz der dahinterliegenden Mauer, Quelle: Amazon (2022), Online-Quelle [21.05.2022].

Abstand und andererseits durch Abdeckungen realisiert werden. Zusätzlich sollen auch konventionelle Schutzvorrichtungen für die dahinter liegende Wand eingesetzt werden. Ein sehr verbreiteter Schutz ist der sogenannte Dartring, wie in Abbildung 24 rot dargestellt. Dieser ist nicht nur vom preislichen Aspekt mit ca. 30 Euro sehr günstig, sondern auch von den Abmessungen her vereinheitlicht.<sup>67</sup>

Zusätzlich kann dieser Dartring in die Konzeptionierung integriert werden, da er einen gegebenen Abstand zur DS vorgibt. Dadurch können Freiheitsgrade definiert werden.

Bei einem gut definierten Lichtschutz, z.B. einer Einhausung, kann zusätzlich die Abschirmung der Kameras bewerkstelligt werden.

### **Fertigung und Zugang zu Komponenten:**

Um das System am Ende für den Amateurspieler zugänglich und umsetzbar zu machen, sollen alle Komponenten und Bauteile entweder leicht erwerbbar oder leicht zu fertigen sein. Zusätzlich soll der preisliche Aspekt nicht außer Acht gelassen und die Materialkosten minimiert werden. Deswegen wird auch auf Rapid-Prototyping gesetzt, da mittlerweile die Verfügbarkeit von bzw. der Zugang zu 3D-Druckern fast für jedermann gegeben und erschwinglich ist. Alle 3D-gedruckten Teile sollen der Arbeit digital beigelegt und beschrieben werden.

## **5.2.2 Anforderungen an die Elektronikkomponenten**

Ein entscheidender Faktor für die Umsetzung des Systems sind die elektronischen Komponenten wie Kameras und EPC. Im Zuge des praktischen Teils sollen die Elektronikkomponenten getestet und gegebenenfalls die Hardware angepasst werden. Zu diesem Zeitpunkt der Arbeit können nur Annahmen bezüglich der Anforderungen getroffen werden.

### **Kameras:**

Die Kameras bilden das Herzstück der Hardwarekomponenten. Bei idealer Auswahl der Hardware können Fehler wie Ungenauigkeit der Auswertung, Fischaugenfehler und Fokussierungsfehler im Vorhinein vermieden werden. Um hier auf die richtige Hardware zurückzugreifen, sollen folgende Mindestanforderungen eingehalten werden:

- Mindestauflösung von 1280x720 Pixel (720p, HD-Ready)
- Bildwiederholungsrate von mindestens von 10 FPS
- Manueller Fokus (Bei Autofokus könnte auf nicht relevante Objekte fokussiert werden.)

---

<sup>67</sup> Vgl. Amazon (2022), Online-Quelle [21.05.2022].

- Keine zusätzlichen Treiber (Einfache Einbindung in Windows oder Linux Systemen)
- H.264 Kodierung
- USB-Anschluss

### **EPC:**

Wie in Kapitel 5 beschrieben, gibt es unterschiedliche EPC, welche zum Einsatz kommen können. Der RP ist der am weitesten verbreitete unter ihnen. Da die Auswertegeschwindigkeit mit der Prozessorleistung zusammenhängt, muss während der Softwareentwicklung auf die Ressourcenverbräuche eingegangen werden. Von den Hardwareanschlüssen muss Folgendes beachtet werden:

- Vier USB-Anschlüsse mit mindestens USB 2.0
- LAN- oder WLAN-Verbindung für die Remoteverbindung

### **5.2.3 Softwareanforderungen**

Im folgenden Kapitel wird auf die Softwareanforderungen eingegangen. Diese sollen die Methoden und Funktionen aufzeigen, die im praktischen Teil entwickelt werden.

#### **Einlesen von Bildern:**

Um einen Auswertealgorithmus zu starten, müssen Bilder von den Kameras in einer gewissen Qualität und Wiederholungsrate eingelesen werden. Auch die Zuordnung der Kameras zu der Position um die DS muss variabel gestaltet werden.

#### **Umwandlung in GSB und BB:**

Wie in Kapitel 3.4 beschrieben, sollten eingelesene Farbbilder in GSB und BB umgewandelt werden, um eine bessere Kantendetektion zu erzielen.

#### **Filtern und Glätten:**

Um noch bessere Extraktion der Kanten zu bekommen, müssen die GSB oder BB zusätzlich gefiltert werden.

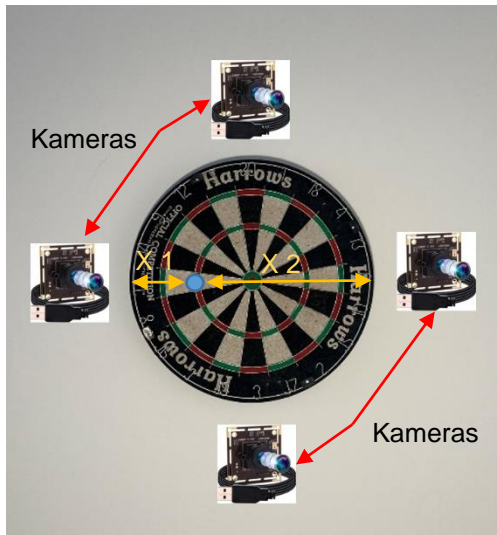


Abbildung 25: Entfernung eines DP zu einer Kamera, Quelle: Eigene Darstellung.

### Detektion von Veränderungen:

Die eingelesenen Bilder sollen fortlaufend vom Kamerasystem aufgenommen werden und auf Veränderungen, in Form eines geworfenen Pfeils, untersucht werden. Diese Funktion soll anschließend den Auswertalgorithmus starten.

### Priorisierung der Kameras:

Um die Probleme bei der Überlappung mehrerer DP und der Ungenauigkeiten der Erfassung von DP, welche eine größere Entfernung zu einer Kamera aufweisen, zu minimieren, muss eine Priorisierung der Kameras vorgenommen werden. Diese soll jeweils zwischen den Kamerapaaren (zwei Kameras für X- und Y-Koordinate) die jeweils optimaleren Kameras auswählen. In der Abbildung 25 sieht man einen geworfenen

DP (blauer Punkt), welcher sich näher der linken Kamera befindet. Hier würde die Auswertung mit der linken Kamera besser funktionieren als mit der rechten Kamera. Wie in Abbildung 25 zu sehen, wird die Priorisierung eines Kamerapaars aufgrund folgender Parameter vorgenommen:

- Entfernung der Kamera zum Dartpfeil
- Überlappung
- Kanten- und Konturerkennung

### Kanten- und Konturendetektion:

Nach dem Detektieren einer Veränderung wird der Detektionsalgorithmus gestartet. Dieser soll den Schaft des DP als Rechteck oder Trapez erkennen und markieren. Zusätzlich erkennt diese Funktion im besten Fall mehrere Objekte. Dies wird in Schritt 2 der Abbildung 26 dargestellt.

### Reduktion auf eine Linie:

Da der Schaft des DP nicht in der DS steckt, sondern deren Spitze, muss eine Funktion entwickelt werden, die aus der extrahierten Trapezkontur eine Linie errechnet, die die Drehachse des DP darstellt. Dies wird in Schritt 3 in der Abbildung 26 dargestellt.

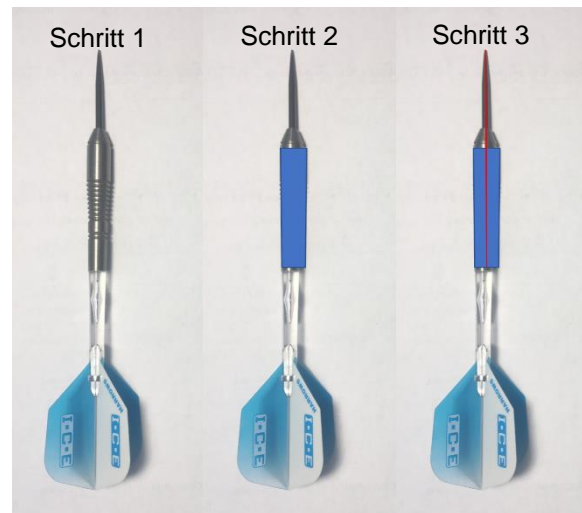


Abbildung 26: Schritte zur Findung der Symmetrieachse eines DP, Quelle: Eigene Darstellung.

### Positionsbestimmung:

Nach der Extraktion der Achse eines DP muss die Position dieser Kontur gefunden werden. Dies soll mittels Koordinatensystems wie in Abbildung 21 erfolgen.

### Zuordnung zu Punkten auf einer DS:

Nach der Positionsbestimmung müssen die ausgelesenen Koordinaten auf die Punkte der einzelnen Segmente der DS zugeordnet werden.

### Erkennung von abgezogenen Pfeilen:

Nach Beendigung eines Wurfvorganges müssen die DP von der DS entfernt werden. Dieser Vorgang muss den Start einer neuen Wurfserie einleiten. Dies kann dadurch erzielt werden, dass entweder die Hand als maximale Veränderung im Bild erkannt wird oder der Algorithmus stoppt, bis alle DP entfernt sind.

## 5.3 Gesamtkonzept

In diesem Kapitel wird das Gesamtkonzept vorgestellt, welches aus den Erkenntnissen des Kapitels 5.2 ausgearbeitet wird.

### 5.3.1 Konzept des Hardwareaufbaus

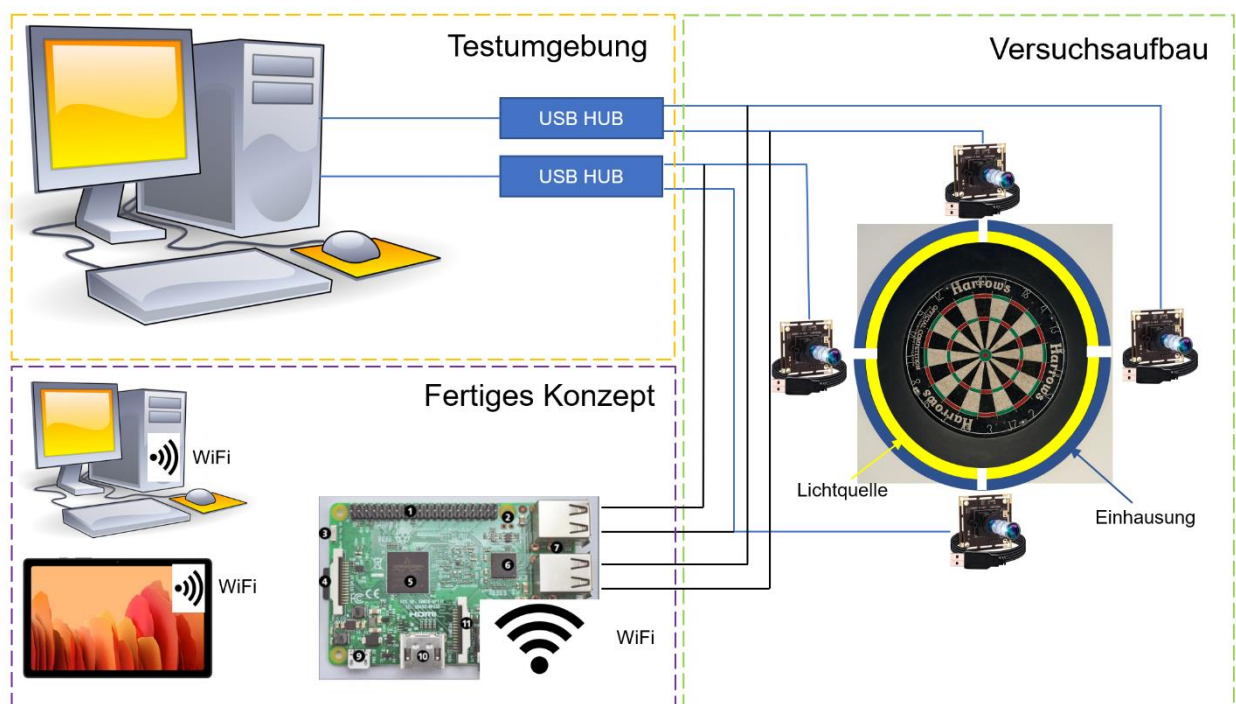


Abbildung 27: Gesamtkonzept für den Hardwareaufbau, Quelle: Eigene Darstellung.

Die Abbildung 27 zeigt das Gesamtkonzept des Hardwareaufbau. Diese ist wie folgt zu verstehen:

### Versuchsaufbau:

Im Versuchsaufbau soll eine Einhausung gefertigt werden, welche die DS komplett umschließt und vier Durchbrüche enthält, die für die Kameras vorgesehen sind. Diese Einhausung soll nicht nur die Kameras



schützen, sondern auch Träger für die konstante Lichtquelle dienen. Zusätzlich fungiert die Einhausung als neutraler Hintergrund, welche den Kameras und der Software eine bessere Detektion der DP ermöglicht. Die gesamte Einhausung soll in einem CAD-Programm geplant und mittels 3D-Druckverfahren hergestellt werden. Bei dem Versuchsaufbau ist darauf zu achten, dass jedes Bauteil so gestaltet werden muss, dass es an dem Dartboard oder der dahinter liegenden Wand angebracht werden kann.

Die vier Kameras werden anfänglich mittels USB-Schnittstelle mit Testumgebung verbunden.

### **Testumgebung:**

Die Testumgebung setzt sich aus zwei USB-Verteilern sowie einem Windowsrechner zusammen. Auf dem Windowsrechner soll mittels der Programmiersprache C# die Testumgebung für den Algorithmus erstellt werden. Diese wird in Visual Studios erarbeitet.

Auf einem EPC befinden sich wie in Kapitel 2 besprochen vier USB-Ports auf der Platine. Da sich jedoch jeweils zwei USB-Anschlüsse einen BUS teilen, kann es hier zu Datenübertragungs- und Stromversorgungsproblemen kommen. Um diese schon in der Testumgebung zu überprüfen, beziehungsweise diesem Problem entgegenzuwirken, werden jeweils zwei USB-Kameras mittels USB-HUB auf einem USB-Port verbunden. Welches Kamerapaar auf einem BUS angeschlossen werden soll, soll während der Entwicklung der Software erarbeitet werden.

### **Fertiges Konzept:**

Das fertige Konzept, beziehungsweise die geschriebene Software, soll schlussendlich auf einem EPC ausgeführt werden. Je nach erarbeitetem Konzept aus der Testumgebung, werden die vier Kameras mittels USB-Anschluss auf die vier Ports des EPC angeschlossen. Die Verbindung zu dem EPC wird mittels Remotedesktopverbindung hergestellt. Die Remotedesktopverbindung ist standardmäßig auf jedem Windowsrechner verfügbar und wird mittels TCP/IP hergestellt. Wie in Kapitel 2 dargestellt, kann der EPC entweder über WiFi oder Lan-Schnittstelle in das jeweilige Netzwerk oder direkt verbunden werden.

## **5.3.2 Softwarekonzept**

Die Software soll anfänglich, wie in Kapitel 5.3.1 dargestellt, in einer Testumgebung erarbeitet werden. Der Ansatz für dieses Konzept wird in diesem Unterkapitel dargestellt. Um die Positionsbestimmung eines DP auf der DS durchführen zu können, muss ein geeigneter Algorithmus geschrieben werden. Dieser setzt sich aus mehreren Grundschritten zusammen, welche in den folgenden Darstellungen illustriert werden:

### 5.3.2.1 Parameter- und Sucheinstellungen

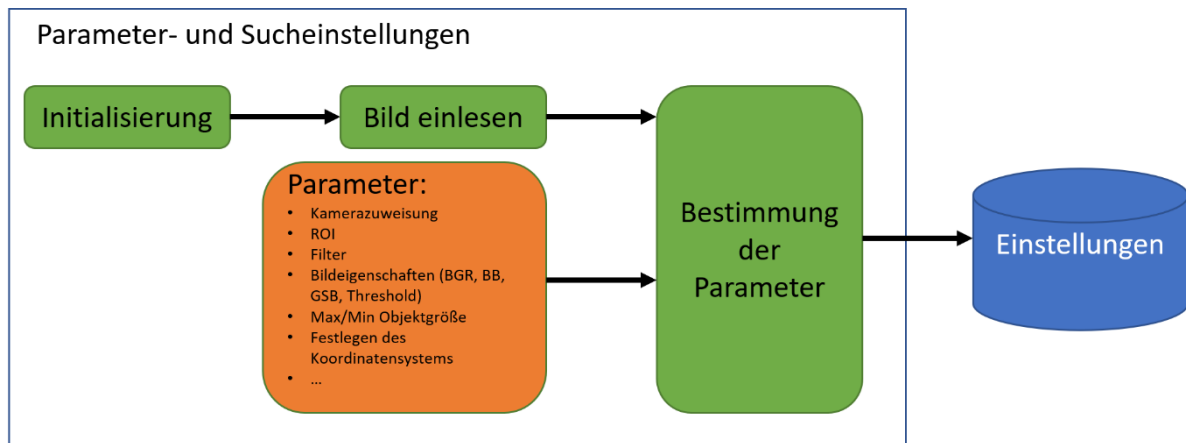


Abbildung 28: Festlegen der Parameter- und Sucheinstellungen, Quelle: Eigene Darstellung.

Da sich das fertige System aus vier Kameras zusammensetzt, müssen vorab Grundeinstellungen für jede einzelne Kamera festgelegt werden, wie in Abbildung 28 dargestellt. Zu Beginn muss die ausgewählte Kamera initialisiert werden, um überhaupt ein Bild der jeweiligen Kamera einlesen zu können. Nachdem ein Bild eingelesen wird, auf dem sich ein DP befindet, wird das Bild auf einer Graphical User Interface (GUI) ausgegeben. Diese GUI soll mehrere Einstellungsmöglichkeiten aufweisen, z.B. Filterarten und Auflösung, die zum Auswerten der DP nötig sind. Nach Abschluss der Parameterfindung und Einstellung der jeweiligen Kamera werden alle Parameter in ein externes Parameterfile gespeichert, welches während des Programmablaufs aufgerufen werden kann. Diese Einstellungen beeinflussen den Grad der Genauigkeit der Positionsfindung eines DP. Jede Kamera wird unterschiedliche Einstellungen aufweisen, da die Positionierung zu den anderen Kameras abweichen wird. Zusätzlich kann dadurch ein modularer Aufbau sichergestellt werden, wodurch z.B. bei einem Kameratausch sehr schnell neue Einstellungen gesetzt werden können. Nach Abschluss der Kameraeinstellung kann der Scan gestartet werden.

### 5.3.2.2 Der Scan

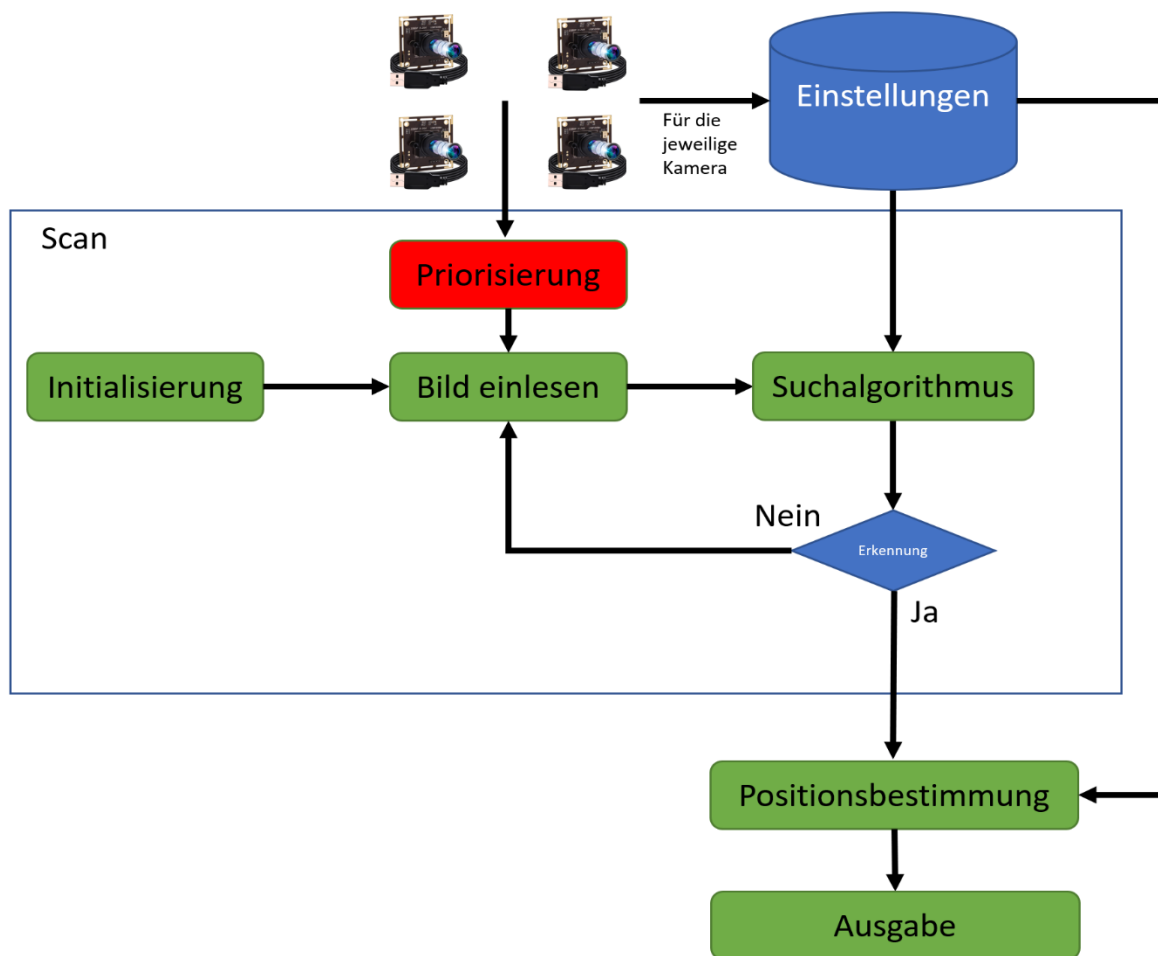


Abbildung 29: Ablauf der Erkennung eines DP, Quelle: Eigene Darstellung.

Zu Beginn des Scans, welcher in Abbildung 29 dargestellt ist, müssen die Kameras wieder initialisiert werden. Im Anschluss wird von einer Kamera Bild für Bild eingelesen und dieses an den Suchalgorithmus weitergegeben. Der Suchalgorithmus nutzt die Einstellungen der jeweiligen Kamera, um das eingelesene Bild weiterzuverarbeiten. Sobald der Suchalgorithmus einen Pfeil erkannt hat, werden die Daten noch einmal durch die Erkennungsschleife geschickt. Dieser zusätzliche Durchlauf wird für die Priorisierung genutzt. Die Priorisierung wählt das geeignete Kamerapaar für die Positionsbestimmung aus. Der Ansatz dieser Priorisierung wird in Kapitel 5.2.3 beschrieben. Die Bilder des jeweiligen Kamerapaars werden erneut an den Suchalgorithmus geschickt. Schlussendlich sollen nur mehr die jeweiligen Koordinaten der Kamera an die Positionierung geschickt werden.

### 5.3.2.3 Der Suchalgorithmus

Im Suchalgorithmus werden die eingelesenen Bilder weiterverarbeitet und die Koordinaten des Dartpfeils errechnet. Dazu wird das Bild von einer Kamera eingelesen und mit den definierten Einstellungen weiterverarbeitet. Der Suchalgorithmus wird gestartet, wenn sich eine gewisse Änderung auf der DS einstellt. Diese Änderungsdetektion erfolgt dadurch, dass die eingelesenen Bilder von einem jeweiligen Referenz- oder Startbild subtrahiert werden. Wenn sich das Bild verändert, so stellt sich ein schwarzes Bild ein. Eine Bildsubtraktion wird durchgeführt, indem eine Subtraktion der jeweiligen Skalarwerte des Pixels



durchgeführt wird. Bei einem BB, welches nur aus den Farben Schwarz (Skalar (0,0,0)) und Weiß (255,255,255) besteht, würde die Subtraktion bei einer Änderung genau an dieser Position weiße Pixel ausgeben. Diese weißen Pixel können anschließend erkannt werden. Wenn man sich nun nur den Schaft eines Dartpfeils ansieht, würden diese weißen Pixel ein gedrehtes Rechteck ergeben. Dieses Rechteck kann detektiert, mittels Kantendetektion genau an die die Form des Dartpfeils angepasst und eine Symmetrielinie gezogen werden. Die Symmetrielinie des Rechtecks bildet auch die Rotationsachse des DP. Wenn man anschließend diese Achse bis zu der DS zieht, kann man die genaue Position der Spitze in der DS errechnen. Diese Schritte werden in den folgenden Abbildungen dargestellt. Die Umwandlung in ein BB wird in diesen Schritten noch nicht berücksichtigt, da sie ein Teil der praktischen Ausarbeitung darstellt.

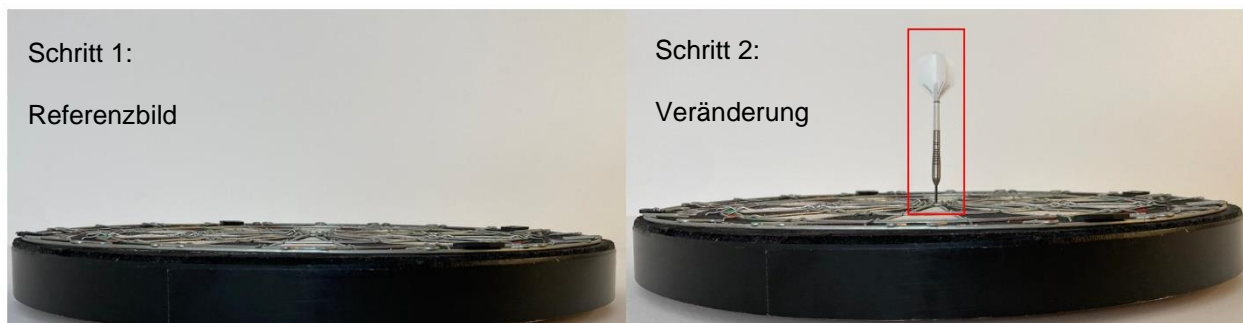


Abbildung 30: Veränderung auf einer DS, Quelle: Eigene Darstellung.

In der Abbildung 30 werden die ersten beiden Schritte dargestellt. Hier wird der Dartpfeil als Veränderung dargestellt.

In der Abbildung 31 werden die Schritte zur Findung der Symmetrielinie beschrieben. Sobald die Pixelkoordinate des DP bestimmt worden ist, können diese Informationen an die Positionierungsfunktion gesendet werden. Um jeweils beide Koordinaten von X und Y zu erhalten, müssen die Schritte 1-6 für jeweils zwei Kameras, welche sich im 90° Winkel befinden, durchgeführt werden.

Sobald die Positionierung durchgeführt worden ist, können die erzielten Punkte des DP ausgegeben werden. Nach Abschluss dieses Vorganges wird das aktuelle Bild als neues Referenzbild deklariert und der Suchvorgang kann neu gestartet werden.

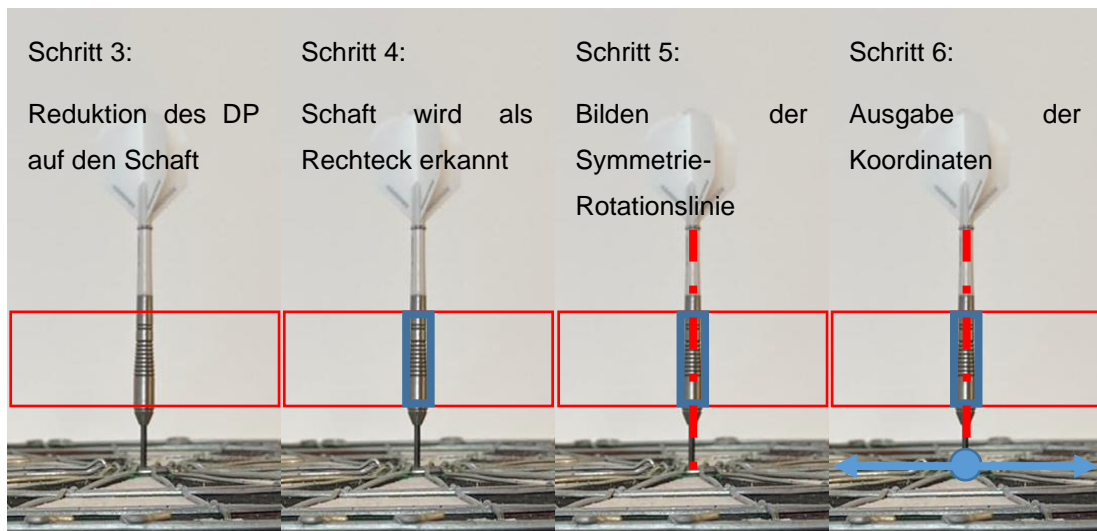


Abbildung 31: Koordinatenbestimmung mittels Reduktion auf Symmetrielinie des DP, Quelle: Eigene Darstellung.

## 6 HARDWAREAUFBAU UND TESTUMGEBUNG

In dem folgenden Kapitel wird auf den Hardwareaufbau und die eingesetzten Komponenten eingegangen. Alle CAD-Daten, welche zum 3D-Druck genutzt werden, werden zusammen mit der erstellten Software auf einem Datenträger bereitgestellt. Des Weiteren wird auf die Testumgebung eingegangen, welche für die Entwicklung des Algorithmus in C# aufgesetzt wird.

### 6.1 Hardwareaufbau und dessen Komponenten

Der Hardwareaufbau besteht aus 7 Komponenten und Komponentengruppen, welche sich aus dem Kapitel 5 ableiten lassen. Diese Komponenten sind:

- Dartboard
- Dartring
- Einhausung
- Kameras (vier Stück)
- Laptop/Standrechner
- LED-Klebestreifen
- USB-HUB (zwei Stück)

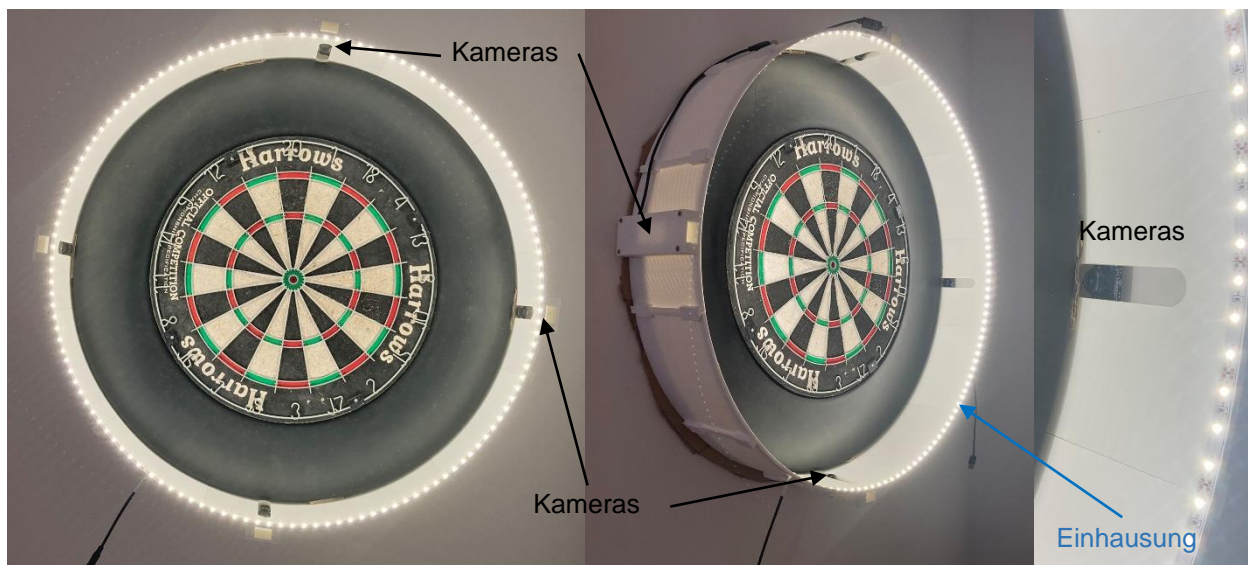


Abbildung 32: Hardwareaufbau der DS, Quelle: Eigene Darstellung.

In der Abbildung 32 ist der Hardwareaufbaus des Dartboards dargestellt. Hier kann man gut erkennen, dass die Einhausung (weißer äußerer Korpus) komplett mit dem Dartring abschließt und diese eine homogene Einheit darstellen. Der Dartring wird nicht an die Wand befestigt, sondern wird vom Dartring und dessen Klemmkraft um die DS gehalten. Der Dartring selbst ist aus elastischem Schaumstoff und weist einen minimal geringeren Durchmesser auf als die DS. Dadurch ergibt sich die Klemmkraft, welche den Dartring und die Einhausung an der DS fixiert, wobei die DS selbst an die Wand geschraubt wird.

Die Einhausung besteht aus 12 Sektoren, die modular ausgetauscht werden können.

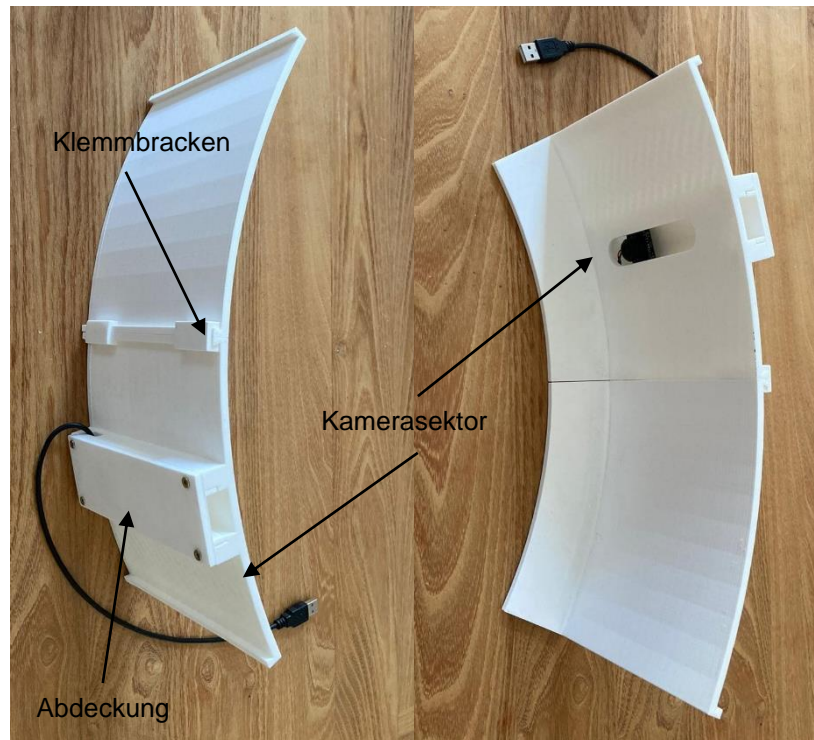


Abbildung 33: Einzelmodule für die Einhausung, Quelle: Eigene Darstellung.

In der Abbildung 33 wird jeweils ein Kamerasektor- und Abdeckungssektormodul dargestellt. Auf der Rückseite des Kamerasektormoduls ist eine verschraubte Abdeckung angebracht, welche für den Verbau der Kamera notwendig ist. Die Abdeckung besitzt einen seitlichen Durchbruch, welcher für das USB-Kabel der Kamera vorgesehen ist.



Abbildung 34: Klemmbracken, Quelle: Eigene Darstellung.

In der Abbildung 34 werden die seitlichen Klemmbracken dargestellt, welche die einzelnen Module zusammenhalten. Die trapezförmige Kontur, welche das Gegenstück an den Modulen aufweist, halten mittels Formschluss die einzelnen Sektoren zusammen. Durch den modularen und formschlüssigen Aufbau können beschädigte Module leicht ausgetauscht werden.

Alle Bauteile, welche für die Einhausung notwendig sind, sind mittels PLA (Polylactid) 3D-Druck-Verfahren hergestellt worden. Dadurch kann der Aufbau schnell und kostengünstig erzeugt werden. Die Druckzeit eines Sets, bestehend aus Einhausungsmodul, Kamerasektormodul, Abdeckung und jeweils vier Klemmbracken, dauert je nach Voreinstellungen bis zu 12 Stunden.



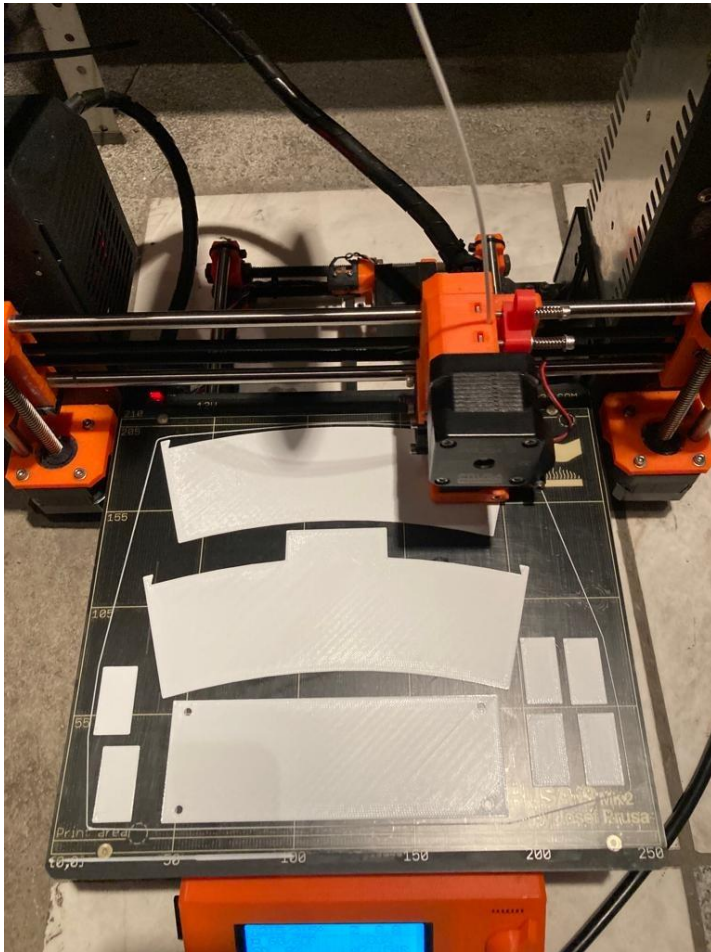


Abbildung 35: 3D-Druck eines kompletten Drucksets der Einhausung, Quelle: Eigene Darstellung.

Alle Bauteile können auf einem Druckbett von 250 mm x 250 mm, welches als Standardmaß gilt, auf einmal gedruckt werden. Alle Druckteile werden mit dieser Arbeit auf der CD mitgeliefert.

In der Abbildung 35 wird dieser 3D-Druck auf einem Prusa i3-Drucker hergestellt. Der komplette Aufbau benötigt ca. 2230 g Filament, was, je nach Hersteller, Kosten von je 50€ erzeugt.

Auf der Einhausung ist die LED-Lichtquelle angebracht, welche für den konstanten Lichteinfluss notwendig ist. Zusätzlich beleuchtet die Lichtquelle die DS soweit, dass keine zusätzliche Lichtquelle notwendig ist. Durch die Dimmungseigenschaft von LEDs kann der Lichteinfluss stufenlos eingestellt werden.



Abbildung 36: LED-Klebestreifen und Durchbruch in der Einhausung, Quelle: Eigene Darstellung.

Der LED-Streifen besitzt auf der Rückseite einen Klebestreifen, der eine einfache Befestigung in der Einhausung ermöglicht.

Dieser LED-Streifen wird in Abbildung 36 dargestellt.

Das Kabel wird durch eine Bohrung in der Einhausung gezogen, um das Kabel vor den metallischen Dartsitzen weitestgehend zu schützen. Dieser Durchbruch wird auch in der Abbildung 36

auf der rechten Bildseite dargestellt. Durch die modulare Bauweise der Einhausung kann der Einhausungssektor mit der Bohrung an einer beliebigen Stelle der DS angebracht werden. Die Kosten für den LED-Streifen betragen je nach Lieferant zwischen 10 und 15 €. Der Vorteil dieser LED-Streifen liegt darin, dass er je nach Bedarf gekürzt und somit genau auf die Größe der Einhausung zugeschnitten werden kann.



Abbildung 38: Kameramodule, Quelle: Amazon (2022), Online-Quelle [16.Mai.2022].



Abbildung 37: USB-Verteiler, Quelle: Eigene Darstellung.

Die eingesetzten Kameramodule, in Abbildung 38 dargestellt, besitzen ein 3 Megapixel Objektiv mit einem Erfassungswinkel von 110°. Die Kameramodule sind das Herzstück des Hardwareaufbaus und sollten deswegen eine Pixelanzahl von 1 MP nicht unterschreiten. Die Kosten für die Module liegen bei 25 € pro Stück. Die 4 Module werden, wie in Abbildung 33 dargestellt, in die Einhausung integriert und sind so vor den DP geschützt. Die Module werden jeweils in einem 90° Winkel zueinander ausgerichtet und sind zur DS-Mittelpunkt ausgerichtet. Die USB-Verbindung zum Standrechner wird über zwei USB-Splitter und USB-Verlängerungen, wie in Abbildung 37 dargestellt, bewerkstelligt. Wie in Kapitel 5.3.1 beschrieben, wird anfänglich mit einem USB-HUB oder USB-Splitter gearbeitet, um etwaige BUS-Probleme, welche am EPC auftreten können, vorab zu vermeiden. Bei der USB-Verlängerung muss darauf geachtet werden, dass bei einer gewissen Länge mit einer aktiven Stromversorgung für die Kameramodule gearbeitet werden muss, um eine konstante Versorgung sicherstellen zu können. Bei einer Länge von 5m sind

bei dem Versuchsaufbau keine Probleme festgestellt worden.

Für den Versuchsaufbau wird ein Standard-Windows-Rechner Windows 10 Education 64-bit verwendet. Die Software wird mit C# unter Visual Studios Enterprise 2019 erarbeitet. Diese wird den Studierenden von der Fachhochschule Campus02 als Lernmittel zur Verfügung gestellt.

## 6.2 Softwaretestumgebung

Wie im Kapitel 5 beschrieben, soll das initiale Softwarekonzept in einer Softwaretestumgebung (SWT) aufgesetzt werden. Hierfür wird eine GUI erstellt, die alle Funktionen der Anforderungsanalyse, wie im Kapitel 5 beschrieben, darstellen und ausgeben soll. Auf Basis dieser Erkenntnisse kann im Anschluss der Algorithmus für die Software aufgesetzt werden. In der Abbildung 39 wird die SWT-Umgebung dargestellt und deren Inhalte beschrieben. Um die Inhalte besser beschreiben zu können, wird die GUI in unterschiedliche Sektoren eingeteilt, welche farblich in der Abbildung dargestellt werden. Diese Sektoren werden im folgenden Unterkapitel einzeln beschrieben und auf die Quelltextausschnitte des jeweiligen Algorithmus wird näher eingegangen. In diesem Kapitel wird größtenteils auf die Abbildung 39 verwiesen, welche die Basis für alle weiteren Abbildungen in diesem Kapitel darstellt.

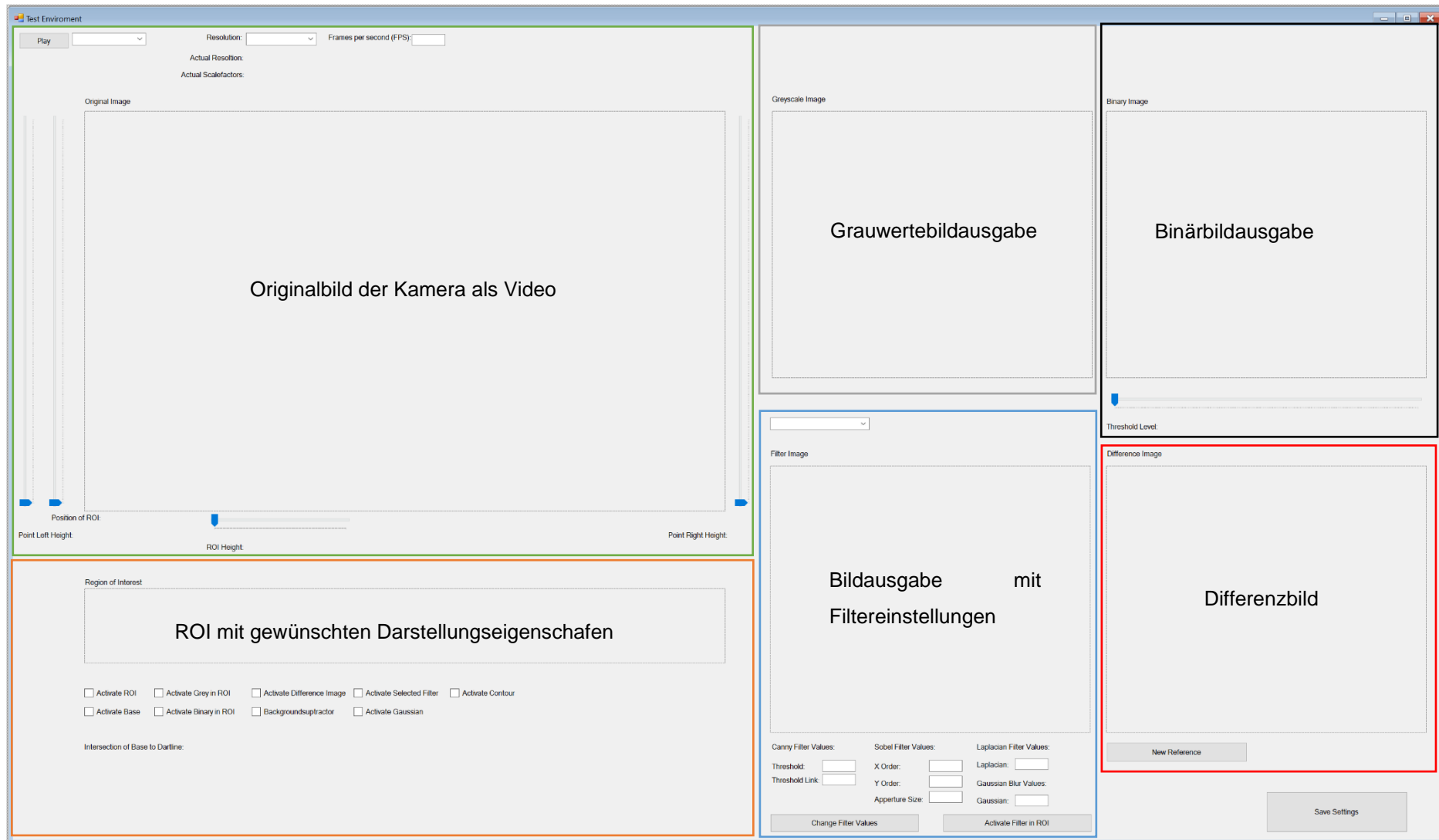


Abbildung 39: GUI der Softwaretestumgebung, Quelle: Eigene Darstellung.

## 6.2.1 Einbinden von ECV in ein C#-Projekt

Um mit ECV arbeiten zu können, muss anfänglich die Bibliothek in das Projekt eingebunden werden. Dies kann am einfachsten mit den Nugent-Paketen, welche von Visual Studios standardmäßig inkludiert sind, durchgeführt werden. Nach Erstellen eines neuen C#-Projektes kann dies, wie in Abbildung 40 dargestellt, durchgeführt werden.

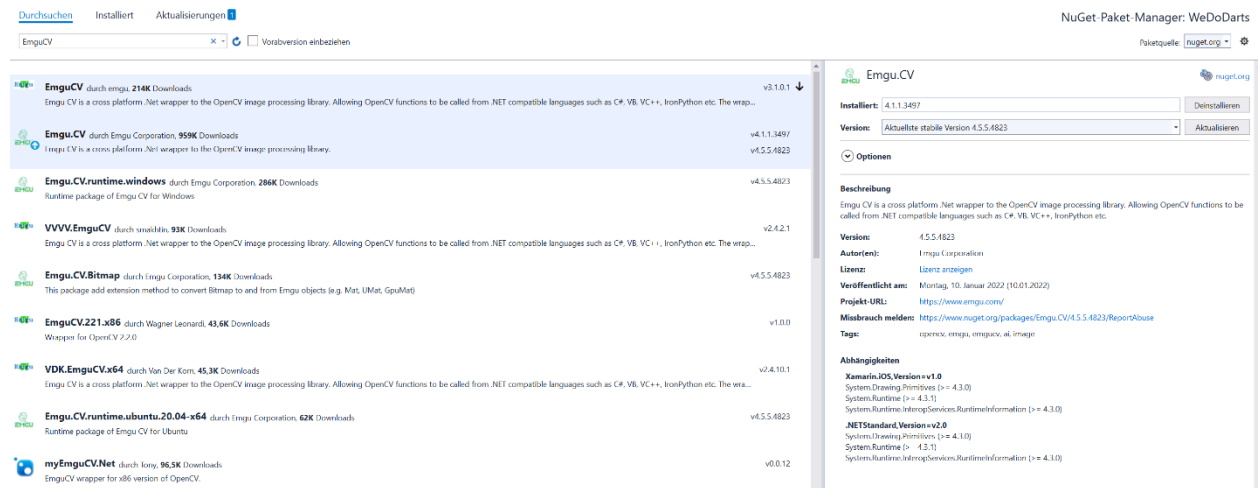


Abbildung 40: Einbinden von ECV in ein C# Projekt, Quelle: Eigene Darstellung.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Emgu.CV;
using Emgu.CV.Structure;
using Emgu.CV.CvEnum;
using Emgu.CV.Util;
```

Um jetzt die gewünschten installierten Pakete verwenden zu können, müssen diese noch per *using* Befehl in die jeweilige Klasse eingefügt werden. (siehe Quelltext 1).

Je nach Versionsart von ECV müssen Befehle unterschiedlich abgerufen werden. Im aktuellen Projekt wird die Version 4.1.1.3497 verwendet.

Quelltext 1: Einbindung von ECV in eine C#-Klasse, Quelle: Eigene Darstellung.



## 6.2.2 Einlesen des Originalbildes

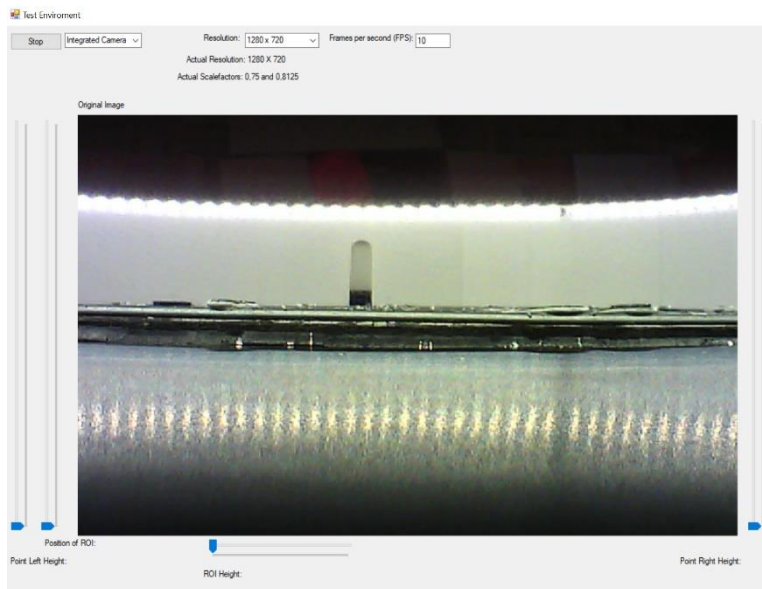


Abbildung 41: Einlesen des Originalbildes einer Kamer, Quelle: Eigene Darstellung.

```
private void Capture_ImageGrabbed(object sender, EventArgs e)
{
    try
    {
        if (capture == null)
            capture = new VideoCapture(selectedcamera, VideoCapture.API.DShow);

        //Mat Object to write Capture from Camera
        Mat m = new Mat();
        capture.Retrieve(m);

        //Save Capture in different Types of Images
        ImgInput = m.ToImage<Bgr, byte>();
        iB_colour_Image.Image = ImgInput;
    }
}
```

Quelltext 2: Quelltextausschnitt zum Einlesen aus von einer Kamer bis zur grafischen Ausgabe, Quelle: Eigene Darstellung.

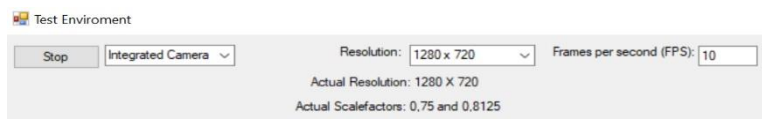


Abbildung 42: Kamera- und Auflösungsauswahl, Quelle: Eigene Darstellung.

des Algorithmus zu testen.

### Probleme und Erkenntnisse:

Da mit dem *VideoCapture()* mittels ganzzahligen Index auf eine Kamera zugegriffen wird, kann es beim Neustart der Software dazu kommen, dass die vom BUS eingelesenen Kameras nicht mehr denselben Index wie beim vorigen Start aufweisen. Diese Problematik zeigt sich daran, dass beim ersten Softwarestart der Index der linken Kamera z.B. bei 1 und bei einem Neustart auf 4 liegt. Dadurch werden indexbezogene

Der grüne Abschnitt in der GUI aus Abbildung 39 repräsentiert das Originalbild, welches von der jeweilig ausgewählten Kamera eingelesen wird. Dieser Abschnitt wird initial verwendet, um die Unterschiede von Auflösung und FPS für den weiteren Verlauf zu testen. Um ein Bild von einer Kamera einlesen zu können, muss anfänglich ein *VideoCapture()* Objekt erstellt werden, um Zugriff auf die Kamera zu bekommen. Hier muss der Index der Kamera, wobei der Index 0 die Referenz zur ersten Kamera darstellt, und die Art des Videoeingangs definiert werden. Im Anschluss muss jedes Bild in ein *Mat()* Objekt überführt werden, welches ein Objekt einer Matrix darstellt, in der die Bildinformationen abgespeichert werden. Je nach Typ, BGR für Farbbild und Gray für GSB, kann die Matrix auf einer Bildvariablen abgelegt werden, welche in einer *ImageBox(iB-colour\_Image)* dargestellt wird. Dieser Vorgang wird in dem in dem Quelltext 2 dargestellt.

Um weitere Einstellungen für das Originalbild vorzunehmen, welches die Grundlage für alle Weiterverarbeitungen ist, können noch Auflösung und FPS angepasst werden, um die Performance der Kameras und

abgespeicherte Kameraeinstellungen auf eine falsche Kamera angewandt. Das Problem wird damit umgangen, dass eine Methode geschrieben wird, welche die jeweilige Hardware-ID der ausgewählten

```
public static string GetCamIDByIndex(int tempindex)
{
    //Function for getting Hardware ID of actual selected Camera
    string tempcamID;
    var allCams = DsDevice.GetDevicesOfCat(FilterCategory.VideoInputDevice).ToList();
    string[] str = new string[allCams.Count];

    var cam = allCams[tempindex];
    tempcamID = cam.DevicePath.Substring(44, 10);

    return tempcamID;
}
```

Kamera ausliest und als Zeichenkette zurückgibt. Diese Methode wird im Quelltext 3 dargestellt. Im Anschluss können alle vorgenommenen Einstellungen auf die Hardware-ID der Kamera gespeichert werden.

Quelltext 3: Funktion zum Auslesen der Kamerahardware-ID mittels Kameraindex, Quelle: Eigene Darstellung.

```
public static int GetCamIndexById(string tempcamId)
{
    //Check the Index of the Camera based on the ID
    var allCams = DsDevice.GetDevicesOfCat(FilterCategory.VideoInputDevice).ToList();
    var index = allCams.FindIndex(x => x.DevicePath.Contains(tempcamId));
    if (index == -1)
    {
        index = 0;
        throw new Exception($"Camera with specified id '{tempcamId}' not found in connected camera devices for Camera#{index}");
    }

    return index;
}
```

Quelltext 4: Methode zum Auslesen des Kameraindex mittels Kamerahardware-ID, Quelle: Eigene Darstellung.

Vice versa kann der Kameraindex über die eingegeben Hardware-ID mit einer eigenen Funktion wieder ausgelesen werden. Diese Funktion wird im Quelltext 4 dargestellt.

### 6.2.3 Erstellung des GSB und BB

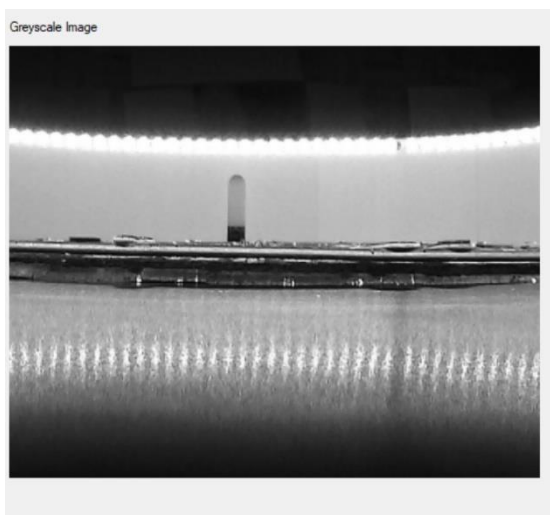


Abbildung 43: Umwandeln des Farbbildes in ein GSB, Quelle: Eigene Darstellung.

Um ressourcenschonender zu arbeiten und Weiterverarbeitungen des Bildes vorzunehmen (z.B. das Bild in ein BB umzuwandeln), wird das Farbbild in ein GSB umgewandelt. Das GSB kann entweder aus dem bestehenden Farbbild konvertiert werden oder direkt von dem anfänglich erstellten *Mat()* Objekt in ein GSB eingelesen werden.

Das eingelesene GSB wird in Abbildung 43 dargestellt.

Das eingelesene GSB wird anschließend in ein BB weiterverarbeitet. Dieses BB muss mit einem gewissen Schwellwert versehen werden, der je nach Eingabewert unterschiedliche Darstellungen eines DP aufweist. Der Schwellwert, englisch Treshold, muss variabel sein, um den passenden Wert für weitere Kantenerkennung zu

finden. Durch einen Schieberegler in der GUI kann der Schwellwert angepasst werden.

Der Quellcode für den variablen Schwellwert wird im Quelltext 5 dargestellt:

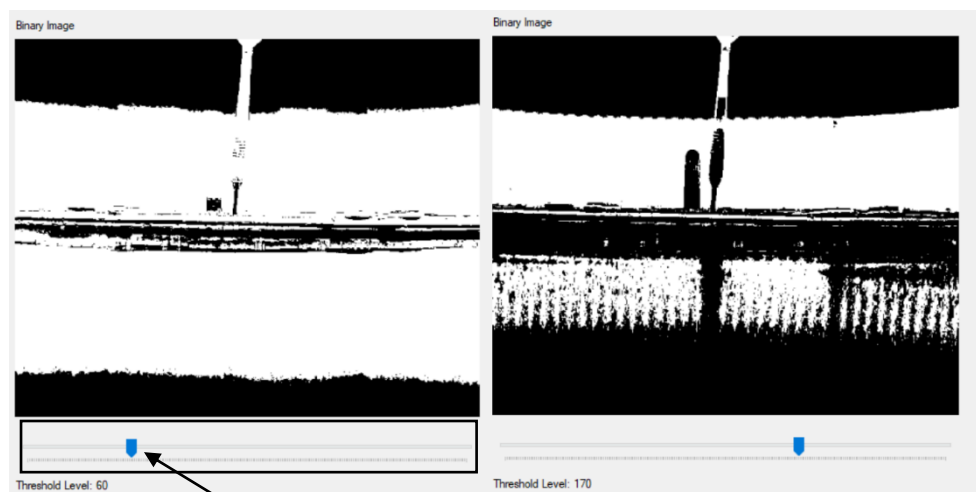
```
//Change Treshold for Binary Picture
ImgBinarizedInput = new Image<Gray, byte>(ImgGrayInput.Width, ImgGrayInput.Height);
double thresholdInput = CvInvoke.Threshold(ImgGrayInput, ImgBinarizedInput, tb_value, 255, Emgu.CV.CvEnum.ThresholdType.Binary);
```

Quelltext 5: Schwellwerteinstellungen für das BB mittels Schieberegler, Quelle: Eigene Darstellung.

Anfänglich wird ein Binärbild mit der gleichen Größe wie das Eingabebild erzeugt. In der darauffolgenden *Threshold()*-Funktion wird das Bild auf den gewünschten Schwellwert *tb\_value* erstellt.

### Probleme und Erkenntnisse:

Wie in der Abbildung 44 dargestellt, liegt der optimale Schwellwert für das BB bei ca. 170. Wie in Abbildung dargestellt, erkennt man den Schaft eines DP bei weißem Hintergrund bei diesem Wert.



Schieberegler für den Schwellwert

Abbildung 44: BB mit unterschiedlichen Schwellwerten, Quelle: Eigene Darstellung.

## 6.2.4 Filtereinstellungen für die Kantendetektion

Um die unterschiedlichen Filterarten zu testen, wird eine eigene Ausgabe für diese Einstellungen auf der GUI erstellt. Diese werden in der Abbildung 39 im grünen Sektor dargestellt. Es wird auf folgende Einstellungen eingegangen:

- Weichzeichnen (Gaussian)
- Der Canny-Filter
- Der Sobel-Filter
- Der Laplace-Filter

Jeder dieser Filtertyp kann separat gesetzt werden. Diese können einfach in ECV aufgerufen und auf GSB oder BB angewandt werden.

```
//Canny Filter
ImgCanny = new Image<Gray, byte>(ImgInput.Width, ImgInput.Height, new Gray(0));
ImgCanny = ImgInput.Canny(canny_TH, canny_TH_Link);

//Sobel Filter
ImgSobel = new Image<Gray, float>(ImgInput.Width, ImgInput.Height, new Gray(0));
ImgSobel = ImgGrayInput.Sobel(sobel_xorder, sobel_yorder, sobel_apertureSize);

//Laplacian Filter
ImgLaplacian = new Image<Gray, float>(ImgInput.Width, ImgInput.Height, new Gray(0));
ImgLaplacian = ImgGrayInput.Laplace(laplacian);

//Gaussian Filter
ImgGaussian = new Image<Gray, byte>(ImgInput.Width, ImgInput.Height, new Gray(0));
ImgGaussian = ImgGrayInput.SmoothGaussian(gaussian);
```

Quelltext 6: Erstellung der Bilder mit den Filtereinstellungen, Quelle: Eigene Darstellung.

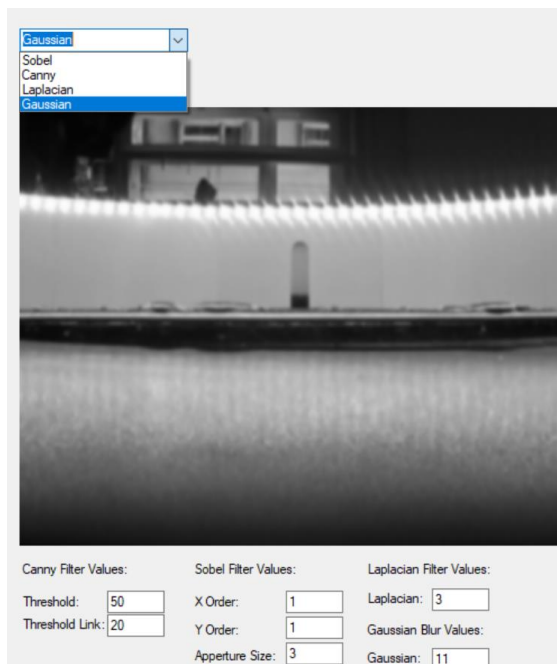


Abbildung 45: Auswahl der unterschiedlichen Filterarten, Quelle: Eigene Darstellung.

Der Quelltext 6 zeigt die Erstellung dieser Bilder. Jeder Filter benötigt unterschiedlich Einstellungsmöglichkeiten. Dies wird in der GUI-Abbildung 39 als blauer Sektor berücksichtigt und die Werte können mittels Eingabetextfelder unterschiedlich gesetzt werden. Im oberen Bereich der Abbildung 45 dargestellt, kann der Filtertyp gewählt und im unteren Bereich der Abbildung gesetzt werden.

Der Weichzeichner ist einer der wichtigsten Filterarten. Durch diesen kann die Extraktion von unterschiedlichen Objekten am einfachsten erzielt werden. Wie im Kapitel 3.4.1.1 beschrieben, können dadurch sich im Vordergrund befindende Objekt besser vom Hintergrund extrahiert werden. Aus diesem Grund wird der Weichzeichner bei jedem Filter als Standard mit hinzugefügt.

### Probleme und Erkenntnisse:

Wie in Abbildung 46 dargestellt, eignet sich der Laplace-Filter am besten für die Kantenextraktion in dieser Anwendung. Beim Sobel-Filter ergibt sich durch den weißen Hintergrund ein fast unleserliche Kantenextraktion, wohingegen der Canny-Filter zu viele Kanten erkennt, was zu einer Störung bei der Erkennung von DP führen kann. Im weiteren Verlauf dieser Arbeit wird daher nur mehr der Laplace-Filter zum Einsatz kommen.

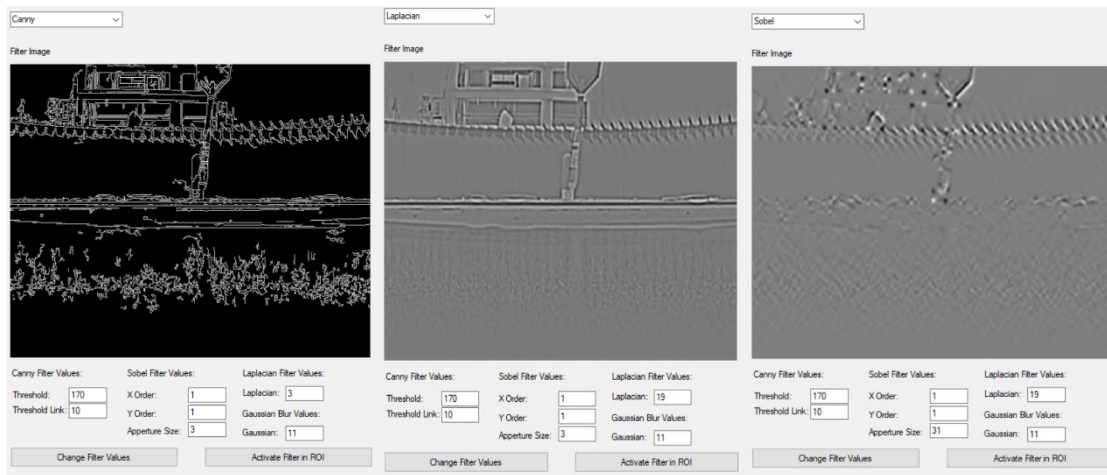


Abbildung 46: Unterschiede der Filtertypen, Quelle: Eigene Darstellung.

### 6.2.5 Anzeigen eines Differenzbildes

In der Abbildung 39 im roten Sektor wird das Differenzbild ausgegeben. Dies dient dazu, den Hintergrund zu subtrahieren und nur mehr die Veränderung im Bild, entstehend durch einen DP, darzustellen. Dadurch können unnötige Informationen entfernt werden. Durch zusätzliches Filtern mit dem Weichzeichner können so auch noch unnötige auftretende Pixel, z.B. durch flackerndes Licht oder kleine Schattierungen, entfernt werden. In ECV kann ein Bild einfach mathematisch subtrahiert werden. Dazu muss anfänglich ein Referenzbild gespeichert werden, welches die leere DS darstellt. Anschließend wird kontinuierlich das neu eingelesene Bild vom Referenzbild Pixelweise subtrahiert. Die Abbildung 47 zeigt auf der linken Seite die anfängliche Subtraktion der beiden Bilder ohne Veränderung auf der DS und auf der rechten Seite, die Bildsubtraktion, sobald ein DP geworfen wird. Auf der linken Seite sieht man sehr gut den Schaft des DP, welcher für die Weiterverarbeitung des Bildes genutzt werden kann.

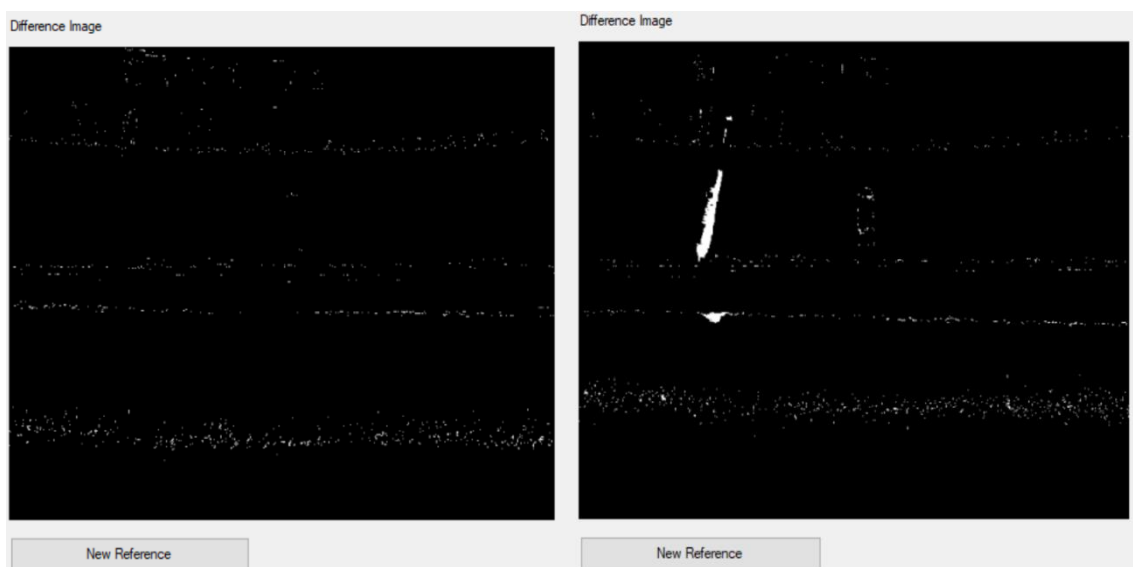


Abbildung 47: Darstellung eines Differenzbildes vor und nach einem DP-Wurfes, Quelle: Eigene Darstellung.

### Probleme und Erkenntnisse:

Wie in der Abbildung 47 erkennbar, werden trotzdem noch viele unnötige Pixel als Differenz dargestellt. Dies kommt größtenteils von sich ändernden Lichteinflüssen, welche durch Spiegelungen entstehen. Um hier ein noch besseres Bild zu bekommen, können diese Pixel mit der *MorphologyEx()*-Funktion entfernt werden. Diese Funktion schließt offene Stellen im Bild automatisch. Dies bedeutet, dass bei einem Schließwert von 10 automatisch alle Stellen, an denen sich eine Ansammlung von maximal 10 weißen Pixel befinden und somit eine Veränderung im Bild erzeugen, zu schwarzen Pixeln umgewandelt werden. Dadurch ergibt sich ein weitaus homogeneres Bild.

## 6.2.6 Minimieren des Bildes auf eine ROI

Um unnötige Informationen eines Bildes zu entfernen, wird das Bild auf eine vordefinierte ROI verkleinert. In dieser ROI werden nun alle gewünschten Parameter berücksichtigt, die aus den Erkenntnissen in diesem Kapitel hervorgegangen sind. Um die ROI zu definieren, wird ein Rechteck in das Originalbild gezeichnet

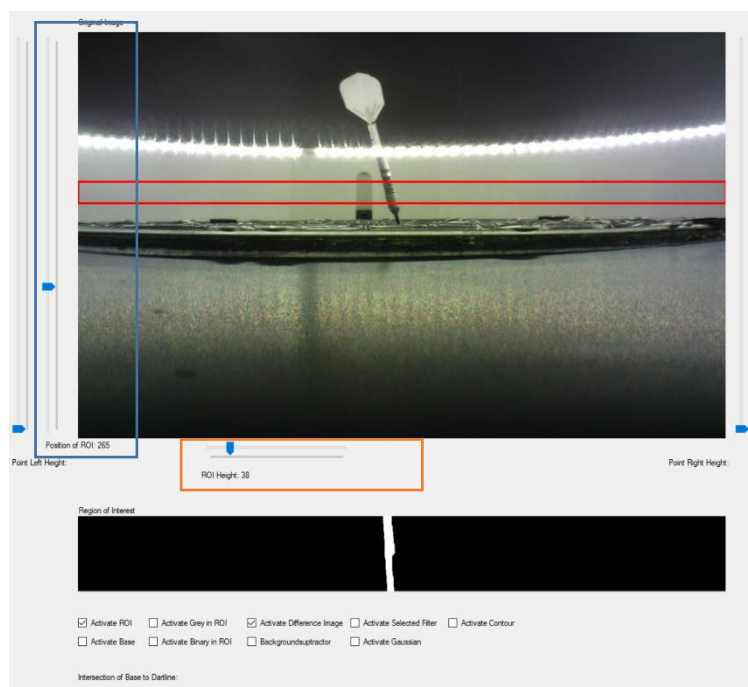


Abbildung 48: Positionierung und Größenänderung der ROI, Quelle: Eigene Darstellung.

und die Position und Höhe mit Schieberegler definiert. Die Abbildung 48 zeigt wie die ROI im Originalbild angepasst wird. Der Schieberegler im blauen Sektor stellt die Position und der Schieberegler im orangen Sektor die Höhe ein. Die ROI kann in ECV durch ein Rechteck definiert und direkt auf ein Bild angewandt werden. Das Eingabebild muss nur mittels der ROI-Methode verarbeitet und auf ein neues Bild gespeichert werden. Dies wird im

```
//ROI of an Grayscale Image
ImgGrayInput.ROI = Rect_ROI_Rezise;
ImgROIGray = ImgGrayInput.Copy();
iB_ROI.Image = ImgROIGray;
```

Quelltext 7: Erstellen einer ROI, Quelle: Eigene Darstellung.

Quelltext 7 beschrieben.

In der Abbildung 48 wird nur aus der definierten ROI ein Differenzbild binär erzeugt und ausgegeben. Man kann nun im unteren Teil der Abbildung sehr gut den Schaft des DP erkennen, welcher auf der DS als Veränderung dargestellt wird.

### Probleme und Erkenntnisse:

Da sich bei Änderung der Bildauflösung auch die Position und Größe der ROI mitverändern muss, wird ein Skalierungsfaktor eingeführt, welcher sich je nach Auflösung ändert. Dadurch kann die ROI immer in derselben Größe und Position im Bild eingebettet werden.



## 6.2.7 Erzeugen eines Rechtecks zum Umspannen eines DP

Wie im vorigen Kapitel besprochen, kann nun mit unterschiedlichster Einstellung versucht werden, ein gedrehtes Rechteck, um den DP in der ROI zu spannen. Anfänglich muss ein Objekt vom Datentyp *Contour()* erzeugt werden. Die aufgerufene Methode *FindContours()* durchsucht das eingelesene Bild nach geschlossenen Konturen einer gewissen Größe. Durch Ablegen der Konturen in einem Variablenfeld *contours [i]*, können diese gespeichert werden. Mittels definierter Abfrage kann nun nach einem gedrehten Rechteck im Bild gesucht werden. Da in der Variable *contours [i]* nicht nur passenden Konturen abgelegt werden, muss abgefragt werden, ob die Kontur eine gewisse Größe *minarea* und ein gewisses Seitenverhältniss *arc* aufweist. Sind beide Abfragen wahr, kann nun das Rechteck in dem Bild, wie in Abbildung 49 dargestellt, eingebettet und zur Weiterverarbeitung genutzt werden. Im Quelltextausschnitt Quelltext 8 wird dieser Ablauf umgesetzt.



Abbildung 49: Umschließen von drei DP mit Rechtecken, Quelle: Eigene Darstellung.

```

ImgContours = ImgROIGray;
CvInvoke.FindContours(ImgROIGray, contours, null, RetrType.External, ChainApproxMethod.ChainApproxSimple);
for (int i = 0; i < contours.Size; i++)
{
    //Bounding Rectangle for evaluating of the Contour Values
    var bbox = CvInvoke.BoundingRectangle(contours[i]);
    var minrect = CvInvoke.MinAreaRect(contours[i]);

    float centerX = minrect.Center.X;
    float centerY = minrect.Center.Y;

    Point Center = new Point((int)centerX, (int)centerY);
    var angle = minrect.Angle+90;

    //Get Cornerpoints and create rotated rectangle
    var minbox = CvInvoke.BoxPoints(minrect);
    Point[] minrectpoints = Array.ConvertAll(minrect.GetVertices(), Point.Round);
    var area = bbox.Width * bbox.Height;
    var arc = (float)bbox.Width / bbox.Height;
    Point P_Center_Endpoint = new Point();

    if (area>minarea && arc > 0.1)
    {
        //Rotate Angle
        if (minrect.Size.Height<minrect.Size.Width)
            angle += 90;
        //Draw Rotated Rectangle in ROI Image
        CvInvoke.PolyLines(ImgROIGray, minrectpoints, true, new MCvScalar(255, 255, 255), 2);
    }
}

```

Quelltext 8: Quelltextausschnitt zur Findung von gedrehten Rechtecken, Quelle: Eigene Darstellung.

### Probleme und Erkenntnisse:

Den größten Einfluss auf die Findung der Kontur hat der Gaussweichzeichner und der dazugehörig Wert. Nach einer Testreihe von 30 DP konnte ein geeigneter Weichzeichenfaktor von 5 evaluiert werden. Mit diesem Wert sind <90% der DP erkannt und mit einem Rechteck umschlossen worden.

## 6.2.8 Schnittpunkterrechnung der Symmetrielinie und Grundebene der DS

Wie im Kapitel 5.3.2.3 besprochen, soll mittels Schnittpunktes der Symmetrielinie des DP und der Grundebene der DS der geworfene Sektor errechnet werden. Hierzu wird anfangs eine Baseline gezogen, welche die Grundeben der DS darstellt. Mittels der ECV-Funktion *Center()* kann der Mittelpunkt eines Rechtecks bestimmt werden. Diese kann nun auf die gefundenen Rechtecke um die DS angewandt werden und eine Symmetrielinie vom Mittelpunkt des Rechtecks parallel zu seinen Zeilen gezogen werden. Der Schnittpunkt zwischen Symmetrielinie und Baseline repräsentiert die Position der DP auf der DS. In der Testumgebung wird hier nur auf eine Kamera eingegangen und es kann somit noch kein geworfener Sektor bestimmt werden. In der Abbildung 50 erkennt man, dass an beiden Bildrändern Schieberegler angebracht sind, die die grüne Baseline im Bild von links nach rechts zeichnen.

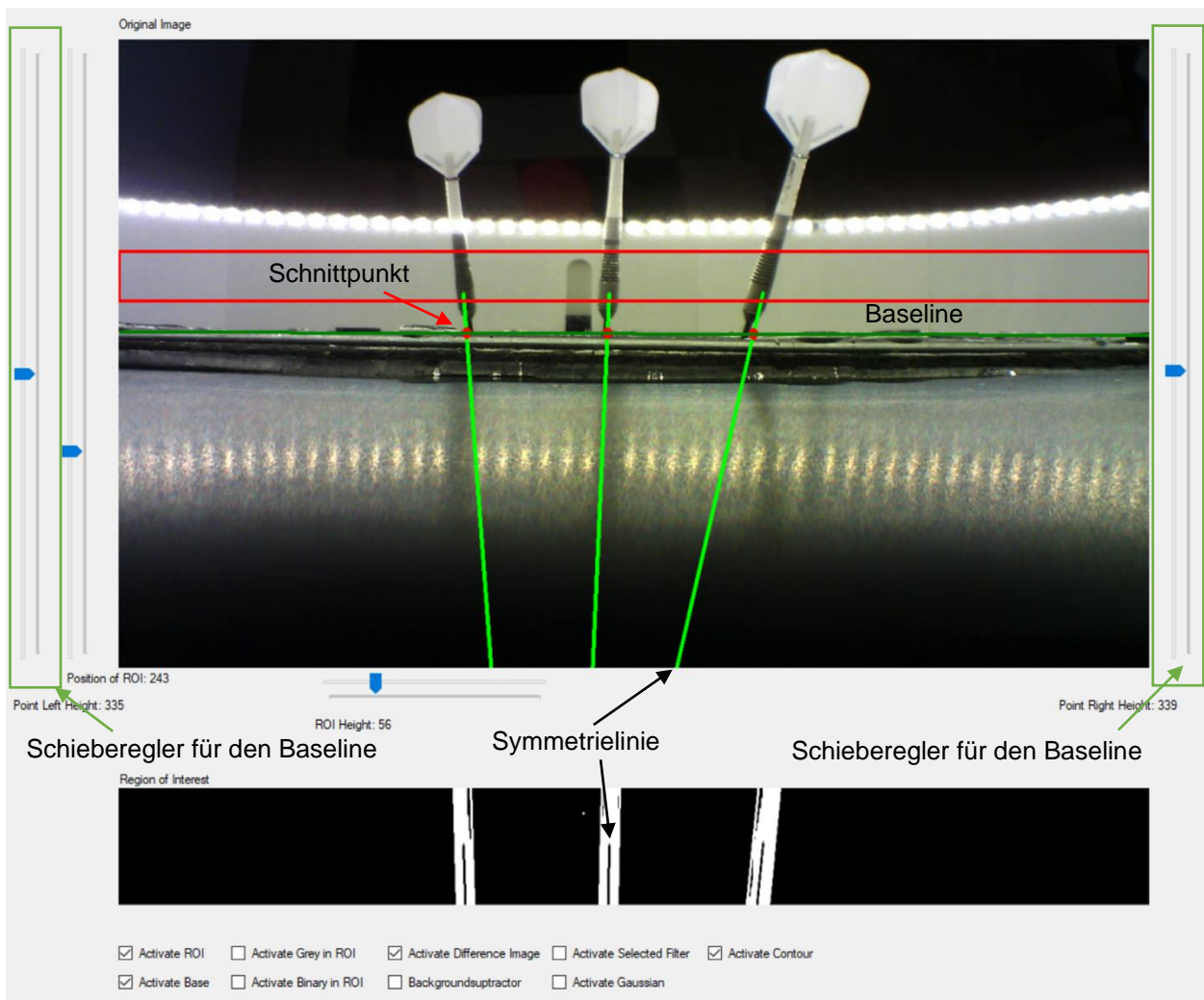


Abbildung 50: Schnittpunktbestimmung der DS und der Symmetrieachse von DP, Quelle: Eigene Darstellung.

Mit der Methode *lineLineIntersection()*, welche im Quelltext 9 beschreiben ist, kann der Schnittpunkt aller DP auf der DS errechnet werden. Diese Punkte repräsentieren die jeweilige X- oder Y-Koordinate auf der DS. Um den Algorithmus zu testen, werden Schnittpunkt, Symmetrielinie und Baseline grafisch im Eingabebild dargestellt. Dies wird in Abbildung 50 dargestellt.



```
public static Point lineLineIntersection(Point A, Point B, Point C, Point D)
{
    // Line AB represented as a1x + b1y = c1
    double a1 = B.Y - A.Y;
    double b1 = A.X - B.X;
    double c1 = a1 * (A.X) + b1 * (A.Y);

    // Line CD represented as a2x + b2y = c2
    double a2 = D.Y - C.Y;
    double b2 = C.X - D.X;
    double c2 = a2 * (C.X) + b2 * (C.Y);

    double determinant = a1 * b2 - a2 * b1;

    if (determinant == 0)
    {
        // The lines are parallel. This is simplified
        // by returning a pair of FLT_MAX
        return new Point(0, 0);
    }
    else
    {
        int x = Convert.ToInt32((b2 * c1 - b1 * c2) / determinant);
        int y = Convert.ToInt32((a1 * c2 - a2 * c1) / determinant);
        return new Point(x, y);
    }
}
```

Quelltext 9: Bestimmung des Schnittpunktes zweier Geraden, Quelle: Eigene Darstellung.

**Probleme und Erkenntnisse:**

Wenn ein DP in einem ungünstigen Winkel auf die DS trifft, werden die umschlossenen Rechtecke auch rotiert. Dadurch kann es vorkommen, dass sich die Seiten der Rechtecke ändern und die Symmetrielinie des Rechtecks um 90° dreht. Um dem entgegenzuwirken, muss eine Abfrage eingebaut werden, die klärt, ob sich die Breite gegenüber der Länge eines Rechteckes längentechnisch immer kürzer verhält. Ist dies nicht der Fall muss die Symmetrielinie auch um 90° gedreht werden, da sich das Rechteck auch gedreht hat. Diese Problematik wird in der Abbildung 51, rechts im Bild, dargestellt.

Die Erkenntnis aus diesem Kapitel ist, dass sich mit diesem Algorithmus drei DP sehr gut erkennen und deren Positionen bestimmen lassen. Dieser Ansatz wird in die finale Umsetzung des Algorithmus einfließen.

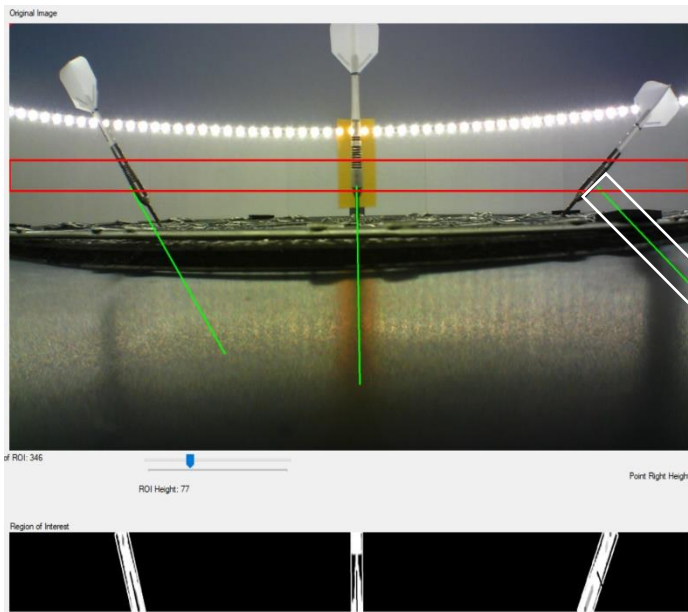
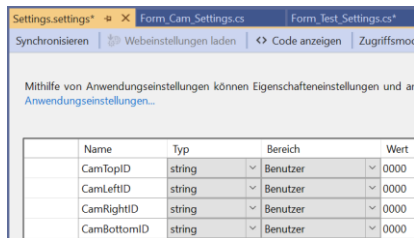


Abbildung 51: Fehlstellung der Symmetrielinie des DP, Quelle: Eigene Darstellung.

## 7 UMSETZUNG DES SOFTWAREALGORITHMUS

Mit den Erkenntnissen aus 6.2 kann nun der Algorithmus und die finale GUI aufgesetzt werden. In diesem Kapitel wird auf die erstellte Software eingegangen. Da der Sourcecode der Software schlussendlich zur Weiterentwicklung öffentlich zugänglich gemacht wird, werden die Kommentare und Quelltextbeschreibung in Englisch verfasst.

### 7.1 Kamerakalibrierung



Name	Typ	Bereich	Wert
CamTopID	string	Benutzer	0000
CamLeftID	string	Benutzer	0000
CamRightID	string	Benutzer	0000
CamBottomID	string	Benutzer	0000

Abbildung 53: Beispiel für Erstellen der Kamer ID Parameter, Quelle: Eigene Darstellung.

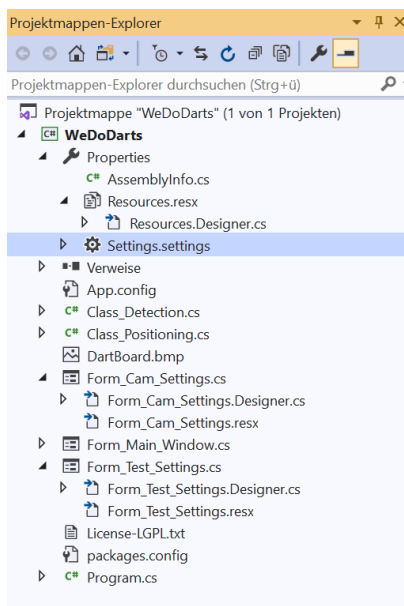


Abbildung 52: Aufrufen der Settings in C#, Quelle: Eigene Darstellung.

Tabelle 7 beschrieben.

Um eine nutzbare Auswertung der Kameras zu bekommen, müssen die Kameras vorab kalibriert und eingestellt werden. Zusätzlich müssen alle eingestellten Information der jeweiligen Kamera abgespeichert werden. Dazu wird eine GUI erstellt, die die notwendigen Einstellparameter zur Verfügung stellt und dem Benutzer die Möglichkeit gibt, die Einstellungen abzuspeichern. Das Abspeichern benutzerspezifischer Parameter kann in den Settings des jeweiligen C#-Projekts durchgeführt werden. In dem Konfigurationsmanager in C# (siehe Abbildung 52) müssen dazu die jeweiligen Parameter erstellt werden. Die Ressourcen können dann global über das Projekt abgerufen und gespeichert werden. Je nach Datentyp können die Parameter abgespeichert werden. Im Beispiel für die Kamera-IDs in Abbildung 53 werden diese als Zeichenkette deklariert. Um nun global auf die Parameter zugreifen zu können, müssen sie mit dem Befehl `CamID = Properties.Settings.Default.CamTopID` auf eine Variable zugewiesen oder mit dem Befehl `Properties.Settings.Default.CamTopID = CamID;` von einer Variable gespeichert werden. In dem ebengenannten Beispiel wird eine Kamera-ID für die obere Kamera abgerufen und anschließend gespeichert.

In der Abbildung 54 wird ein Beispiel dargestellt, wie die Kamerakalibrierung für die linke Kamera aussieht. Hierfür werden die Einstellungen genommen, welche für den Spielbetrieb notwendig sind. Alle wichtigen Einstellparameter werden in den Tabelle 6 und

Camera Settings

Input Image

HBVCAM camera Stop

Actual CamID: 7&2fe25419 <<<< Write new ID

Change Camera

Get Camera ID

Camera Position: Left (3/19)

Resolution: 1280 x 720

Smooth Gaussian: 11 11

Threshold Level: 100

Base Position Y: 339

ROI Height: 30

ROI Position Y: 265

New Reference ABS Diff: 149965

ROI Image

Left Outer Ring: 148

Center of Darboard: 656

Right Outer Ring: 1176

Load Settings

Save Settings

Cancel

Kameraauswahl

Positionsauswahl

Bildauflösung

Textfeld: Smooth Gaussian

Abbildung 54: Beispiel zum Einstellen der Kamerakalibrierung für die linke Kamera, Quelle: Eigene Darstellung.

<b>Parameter, Element oder Funktion</b>	<b>Beschreibung</b>	<b>Rückgabewert (Nur bei Funktionen)</b>
<b>Dropdownmenü: Kameraauswahl</b>	Dropdownmenü zur Auswahl angeschlossenen Kameras.	Kameraindex als Ganzzahl
<b>Dropdown: Positionsauswahl</b>	Wechselt die Position der Kamera um die Dartscheibe.	Kameraposition als Ganzzahl
<b>Button: Get Camera ID</b>	Führt die Funktion aus, welche die Kamera ID der ausgewählten Kamera in das Textfeld schreibt.	Kamera ID als Zeichenkette
<b>Button: Change Camera</b>	Ändert die Kamera ID und speichert sie in eine temporäre Zeichenkettenvariable.	
<b>Dropdownmenü: Resolution</b>	Ändert die Auflösung der gewählten Kamera.	
<b>Textfeld: Smooth Gaussian</b>	Eingabefeld für den Wert des Weichzeichners, wird mit drücken der Entertaste bestätigt.	Weichzeichenwert als Ganzzahl
<b>Trackbar: Threshold</b>	Ändert den Schwellwert für das BB.	Schwellwert als Ganzzahl
<b>Trackbar: Base Position Y</b>	Ändert die Position der Baseline. Wird für die Positionsbestimmung verwendet.	Position als Ganzzahl
<b>Trackbar: ROI Hight Y</b>	Ändert die Position der ROI.	Position als Ganzzahl
<b>Trackbar: ROI Height</b>	Ändert die Höhe der ROI.	Höhe als Ganzzahl
<b>Trackbar: Left outer Ring</b>	Ändert die Grenze der Sektoren am Dartboard. Position, wo sich die linke Grenze des zählbaren Bereichs der DS befindet. Wird für die Positionsbestimmung verwendet.	Position als Ganzzahl

Tabelle 6: Beschreibung der Einstellparameter, Quelle: Eigene Darstellung.

<b>Parameter, Element oder Funktion</b>	<b>Beschreibung</b>	<b>Rückgabewert (Nur bei Funktionen)</b>
<b>Trackbar: Right outer Ring</b>	Ändert die Grenze der Sektoren am Dartboard. Position, wo sich die rechte Grenze des zählbaren Bereichs der DS befindet. Wird für die Positionsbestimmung verwendet.	Position als Ganzzahl
<b>Trackbar: Center</b>	Mittelpunkt der DS. Wird für die Positionsbestimmung verwendet.	Position als Ganzzahl als
<b>Button: New Reference</b>	Erstellt ein neues Referenzbild für das Differenzbild.	
<b>Button: Load Settings</b>	Ladet alle abgespeicherten Parameter für die jeweilige Kamer nach Positionierung.	Variablen werden zugewiesen
<b>Button: Save Settings</b>	Speichert alle Parameter für die jeweilige Kamera nach Positionierung.	Parameterliste als List-Objekt
<b>Label: ABS Diff:</b>	Zeigt den aktuellen Unterschied zwischen Eingabebild und Referenzbild in Pixel an.	

Tabelle 7: Fortsetzung von Tabelle 6, Quelle: Eigene Darstellung.

## 7.2 Der Positionierungsalgorithmus

Um die exakte Position eines Dartpfeils auf der Dartscheibe zu bestimmen, wird ein Algorithmus erarbeitet, der den errechneten Schnittpunkt zwischen Symmetrielinie des DP und Baseline in das Koordinatensystem auf der DS umrechnet.

### 7.2.1 Trichterförmige Erfassung von Bildern von einer Kamera

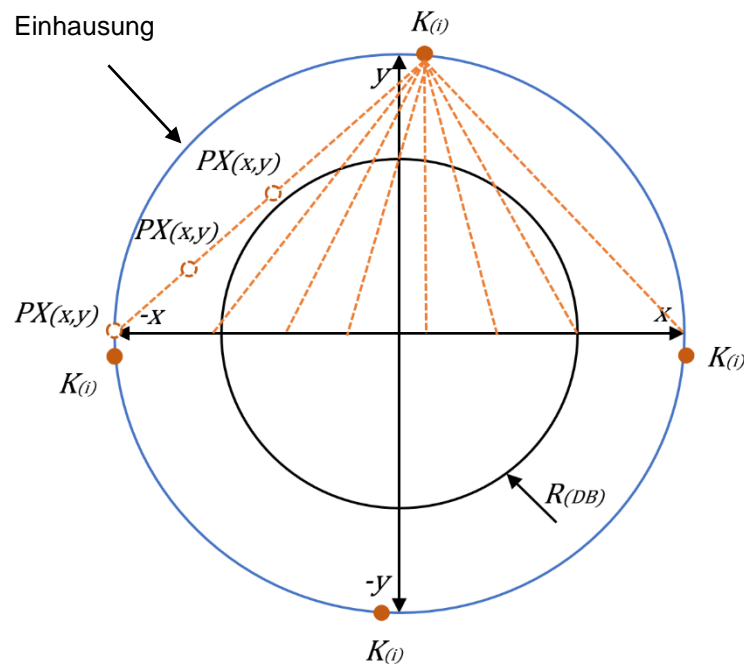


Abbildung 55: Trichterförmiges Erfassen einer Kamera, Quelle Eigne Darstellung.

Die Herausforderung bei diesem Algorithmus ist die trichterförmige Erfassung der Kamera eines Pixels im Raum. Um diese Problematik besser erklären zu können, wird sie in Abbildung 55 illustriert.

Eine Kamera ( $K(i)$ ) ist in der Einhausung, dargestellt als blauer Kreis, eingebaut. Durch Fertigungsfehler wird diese nie exakt auf der Position der Koordinatenachse liegen. Die Kamera erfasst pixelweise das Bild und erkennt je nach Entfernung des Objektes unterschiedliche Koordinatenwerte des Pixels. Das bedeutet, der Kamera ist nicht bewusst in welcher Entfernung das Objekt liegt. Dadurch kann das Objekt überall auf der Geraden des Pixels zur Kamera liegen. Dies wird in Abbildung 55 als  $PX(x,y)$  dargestellt. Dieser Punkt  $PX(x,y)$  wird als Beispiel drei Mal auf der geraden eingezeichnet. Um dieses Problem lösen zu können, wird mittels Schnittpunkterrechnung zweier Geraden gearbeitet.

### 7.2.2 Berechnung des Punktes eines Dartfeils in einem Koordinatensystem

Um das Problem des trichterförmigen Erkennens von Kameras zu nutzen, wird in diesem Unterkapitel auf die Einzelschritte des Algorithmus eingegangen. In dem folgenden Beispiel wird auf nur zwei Kameras eingegangen, da die Auswertung nur zwei Kameras benötigt.

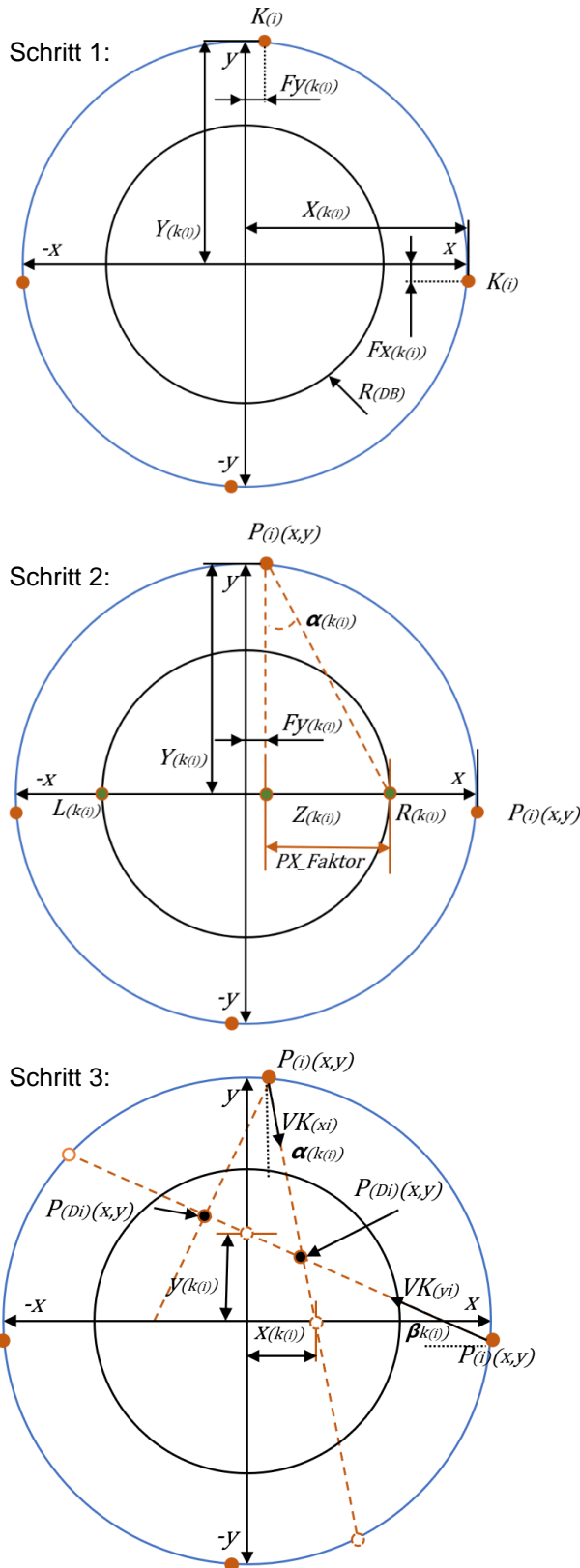


Abbildung 56: Schrittweise Erläuterung der Schnittpunktberechnung, Quelle: Eigene Darstellung.

eingegangen, da die Auswertung nur zwei Kameras benötigt.

#### Schritt 1:

Zu Beginn werden alle bekannten Variablen erhoben um die Kamera  $K(i)$  im Raum zu definieren. Die Entfernung der Kamera zum Mittelpunkt  $Y_{k(i)}$  wird mittels simplen Ausmessens am Aufbau selbst durchgeführt. In der Abbildung 54 aus Kapitel 7.1 kann man erkennen, dass während der Kamerakalibrierung der Mittelpunkt der Kamera optisch ausgewertet wird. Dadurch kann man je nach Kamera den Fehler als  $F_{y(k(i))}$  oder  $F_{X(k(i))}$  definieren.

#### Schritt 2:

Zusätzlich können aus der Kamerakalibrierung die beiden Endpunkte  $L(k(i))$  und  $R(k(i))$  der DS optisch vermessen werden. Für die obere Kamera bedeutet das Folgendes: Wenn man den Fehler  $F_{y(k(i))}$  mit dem Abstand zwischen  $R(k(i))$  und  $Z(k(i))$  addiert, erhält man die Pixelanzahl zwischen Mittelpunkt und rechten Ende der DS. Da die DS durch einen genormten Durchmesser definiert ist, kann so das Verhältnis zwischen Pixel zu Millimeter errechnet werden. Dieser Pixelfaktor,  $PX\_Faktor$ , ist nur für die Entfernung gültig, welche genau die Symmetrieachse  $X$  wiedergibt. Zusätzlich ergibt sich durch die Winkelfunktionen der Winkel  $\alpha(k(i))$ . Mit dem Pixelfaktor, den Positionierungsfehlern und dem ausgemessenen Abstand der Kamera kann man nun die jeweilige Kamera als Punkt  $P(i)(x,y)$  im Raum definieren.

#### Schritt 3:

Bei einem detektierten DP  $P(Di)(x,y)$  im Raum kann man nun den Punkt auf die Ebene der Symmetrieachsen schieben. Durch den vorher



errechneten Pixelfaktor errechnet man die Abstände zu den Symmetrieachsen  $y^{(k(i))}$  und  $x^{(k(i))}$ . Mit diesen Werten schließt man wieder zurück auf die Winkel  $\alpha^{(k(i))}$  und  $\beta^{(k(i))}$ . Mit dem Punkt der Kamera  $P^{(i)}(x,y)$  und dem jeweiligen Winkel wird nun eine Linie gezogen, welche bei der Einhausung enden sollte. Dadurch ergeben sich zwei Linien der jeweiligen Kameras, von denen der Schnittpunkt errechnet wird. Der Schnittpunkt spiegelt den Punkt  $P^{(Di)}(x,y)$  des DP im Raum.

### Probleme und Erkenntnisse:

Je nach geworfenen Quadranten auf der DS und Auswahl der Kamera ändert sich natürlich auch die Richtung, in welche die Linie gezogen werden muss. Z.B. muss bei Detektion der oberen Kamera die Linie im ersten Quadranten in die positive Y- und negative X-Richtung gezogen werden. Im Gegensatz dazu muss bei Detektion mit der unteren Kamera die Linie in positive Y- und positive X-Richtung gezogen werden. Dadurch ergeben sich Abfragen, die im Algorithmus umgesetzt werden müssen. Eine weitere Problematik ergibt sich aus der Bildaufnahme der Kamera. Die Pixel eines Bildes sind von links nach rechts aufsteigend. Dadurch muss auf das Vorzeichen geachtet werden. Wie im vorigen Beispiel bedeutet das, dass die obere Kamera einen positiven X-Wert darstellt, die Position jedoch mit aufsteigendem Wert immer weiter in den negativen X-Teil des Koordinatensystems rutscht. Bei Detektion mit der unteren Kamera wandert der steigende X-Wert des Pixels immer weiter in die positive Richtung des X-Wertes des Koordinatensystems.

Diese Abfragen werden im Quelltext 10 dargestellt.

```
//Generate Point outside of the Dartboard for calculating the intersection
//If the horizontal Camera is on the top (Index 0) and the x value is positiv, The Dart must be in the second Quarter
//x value = Center of the camera in pixel - detected x value of the dart
if (campos_horizontal == 0 && x_value > 0)
    P_new_hor = new Point(Convert.ToInt32(P_cam_hor.X - Math.Sin(alpha) * 10000), Convert.ToInt32(P_cam_hor.Y - Math.Cos(alpha) * 10000));
else if (campos_horizontal == 0 && x_value < 0)
    P_new_hor = new Point(Convert.ToInt32(P_cam_hor.X + Math.Sin(alpha) * 10000), Convert.ToInt32(P_cam_hor.Y - Math.Cos(alpha) * 10000));
else if (campos_horizontal == 2 && x_value > 0)
    P_new_hor = new Point(Convert.ToInt32(P_cam_hor.X + Math.Sin(alpha) * 10000), Convert.ToInt32(P_cam_hor.Y + Math.Cos(alpha) * 10000));
else
    P_new_hor = new Point(Convert.ToInt32(P_cam_hor.X - Math.Sin(alpha) * 10000), Convert.ToInt32(P_cam_hor.Y + Math.Cos(alpha) * 10000));

if (campos_vertical == 1 && y_value > 0)
    P_new_ver = new Point(Convert.ToInt32(P_cam_ver.X - Math.Cos(beta) * 10000), Convert.ToInt32(P_cam_ver.Y + Math.Sin(beta) * 10000));
else if (campos_vertical == 1 && y_value < 0)
    P_new_ver = new Point(Convert.ToInt32(P_cam_ver.X + Math.Cos(beta) * 10000), Convert.ToInt32(P_cam_ver.Y - Math.Sin(beta) * 10000));
else if (campos_vertical == 3 && y_value > 0)
    P_new_ver = new Point(Convert.ToInt32(P_cam_ver.X - Math.Cos(beta) * 10000), Convert.ToInt32(P_cam_ver.Y + Math.Sin(beta) * 10000));
else
    P_new_ver = new Point(Convert.ToInt32(P_cam_ver.X + Math.Cos(beta) * 10000), Convert.ToInt32(P_cam_ver.Y + Math.Sin(beta) * 10000));
```

Quelltext 10: Abfrage, in welchen Quadranten sich der DP befindet, und Ziehen der Linie zur Berechnung des Schnittpunktes, Quelle: Eigene Darstellung.

## 7.2.3 Punktebestimmung auf der Dartscheibe

Wenn man den errechneten Punkt  $P^{(Di)}(x,y)$  aus Kapitel 7.2.2 auf die DS umlegt, wird nun auf die erzielten Punkte umgerechnet. Jeder Sektor in der DS besitzt einen Winkel von 18°. Der Radius und der Winkel werden durch die Koordinaten des Punktes mittels Winkelfunktionen bestimmt. Durch Abfrage im Algorithmus, in welchem Sektor und welchem Radius sich der DP, befindet, kann so auf die Punkte zurückgerechnet werden.

Dieses Vorgehen wird in Abbildung 57 dargestellt.



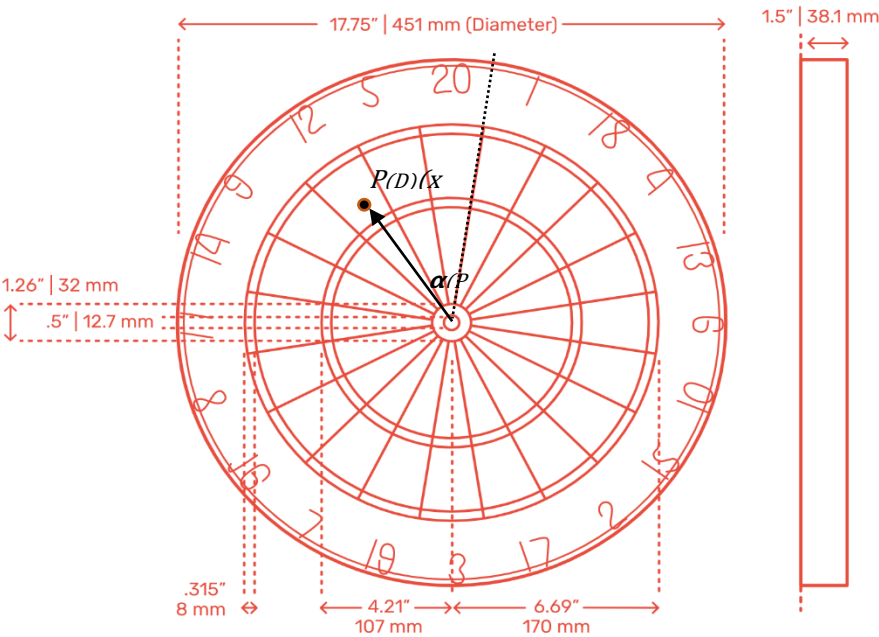


Abbildung 57: Abmessungen eines Dartboards, Quelle: Dimensions (2021), Online-Quelle [21.05.2022] (leicht modifiziert).

## 7.2.4 Der Detektionsalgorithmus

Auf Basis der Erkenntnisse aus dem vorigen Kapitel kann nun der Detektionsalgorithmus definiert werden.

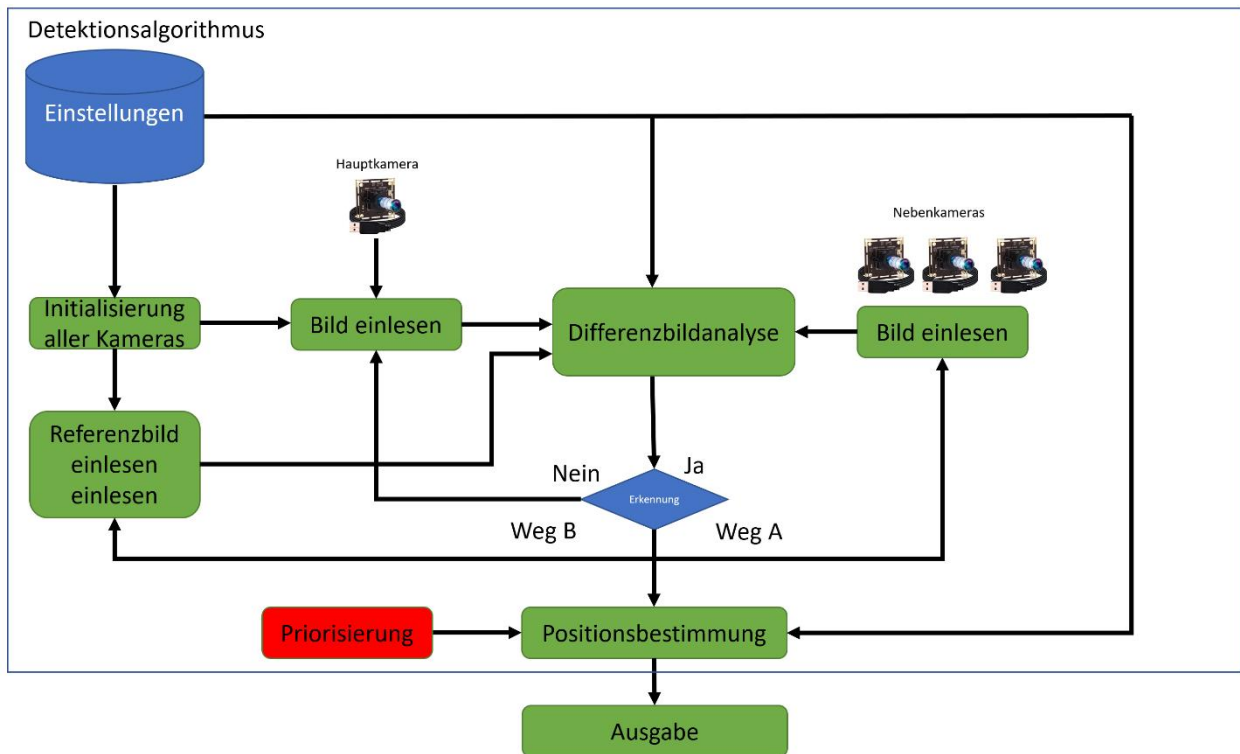


Abbildung 58: Ablauf des Detektionsalgorithmus, Quelle: Eigene Darstellung.

Dieser wird in einer separaten C#-Klasse aufgesetzt, um je nach Anwendung darauf zugreifen zu können. In dem folgenden Kapitel wird mit Hilfe von Ausschnitten von Quelltexten auf die Umsetzung eingegangen.

In der Abbildung 58 wird der Ablauf der Detektion dargestellt. Der Ablauf wird in unterschiedlichen Phasen beschrieben.

### Initialisierungsphase:

In der Initialisierungsphase werden alle Kameraeinstellungen in Schleifen aus der Einstellungsdatei geladen. Es werden die einzelnen *VideoCapture()*-Objekte mit der jeweiligen Kameraeinstellung erzeugt. Nach Erstellung der Objekte wird sofort das erste Referenzbild, welches die DS allein darstellt, aufgenommen. In den folgenden Quelltexten wird dieser Vorgang beschrieben:

```

//Create VideoCapture Objects for all four cameras
for (int i = 0; i < 4; i++)
{
    //Load all Setting for the detection
    DetectionLoadAllSettings(i);
    capture[i] = new VideoCapture(Program.GetCamIndexById(CamID), VideoCapture.API.DShow);
    capture[i].SetCaptureProperty(CapProp.Fps, FPS);
    capture[i].SetCaptureProperty(CapProp.FrameWidth, reswidth);
    capture[i].SetCaptureProperty(CapProp.FrameHeight, resheight);
    capture_values[i, 0] = reswidth;
    capture_values[i, 1] = resheight;
    capture_values[i, 2] = gaussianlevel;
    capture_values[i, 3] = thresholdlevel;
    capture_values[i, 4] = ROIheight;
    capture_values[i, 5] = ROIpos;
    capture_values[i, 6] = BasePos;
    capture_values[i, 7] = Program.GetCamIndexById(CamID);
    rectangle[i] = new Rectangle(0, ROIpos, reswidth, ROIheight);
    //Take a Reference Picture
    Take_Reference_Image(i);
}

```

Quelltext 11: Erstellung der *VideoCapture()* Objekte mit den jeweiligen Kameraeinstellungen, Quelle: Eigene Darstellung.

```

private void Take_Reference_Image(int i)
{
    //Skip first three frames
    for (int j = 0; j < 4; j++)
    {
        if (capture[i] == null)
            capture[i] = new VideoCapture(capture_values[i, 7], VideoCapture.API.DShow);
        Input[i] = capture[i].QueryFrame().ToImage<Gray, byte>();
    }
    //Generate ROI out of setting Information
    Reference[i] = MakeROIinImage(Input[i], rectangle[i]);
    Binary_Reference[i] = new Image<Gray, byte>(Reference[i].Width, Reference[i].Height);
    //Generate Binary Image
    Binary_Reference[i] = MakeBinaryImage(Reference[i], capture_values[i, 3], capture_values[i, 2]);
}

```

Quelltext 12: Referenzbild einlesen für alle Kameras, Quelle: Eigene Darstellung.

### Kontinuierliches Einlesen von einer Hauptkamera.

In dieser Phase werden mit einer definierten Framerate von einer Hauptkamera kontinuierlich Bilder eingelesen und in das Imagearray *Input[]* abgelegt. Es wird deswegen nicht gleichzeitig von allen Kameras eingelesen, um die Recherauslastung zu minimieren. Das eingelesene Bild wird kontinuierlich mit dem Referenzbild verglichen. Dabei wird der absolute Unterschied der Bilder ermittelt. Die Schleife wird so lange durchlaufen, bis ein Überschreiten eines definierten Schwellwertes des absoluten Unterschiedes der Bilder geschieht. Diese bedeutet, dass eine Veränderung auf der DS durchgeführt und ein geworfener DP auf der Scheibe erkannt worden ist. Während des ganzen Ablaufs werden die Bilder mit den Einstellungen aus der Kamerakalibrierung verarbeitet. Dazu werden BB erzeugt, Filter angewandt und die ROI erzeugt.

### Einlesen der Nebenkameras:

Sobald eine Detektion erfolgt, werden die restlichen Bilder der der Nebenkameras eingelesen und eine Differenzanalyse durchgeführt. Das Referenzbild der Kameras ist anfangs beim Referenzieren der

Kameras eingelesen und auf das Imagearray *Reference[]* abgelegt worden. Dieser Ablauf ist in der Abbildung 58 mit Weg A beschrieben und wird vor dem Weg B durchgeführt. Sobald alle eingelesenen Bilder mit den dazugehörigen Referenzbildern verglichen worden sind, werden die eingelesenen Bilder als neue Referenzbilder abgelegt. Dadurch werden schon vorhanden DP im Algorithmus nicht mehr berücksichtigt.

Der folgende Quelltext 13 beschreibt den Schritt „Kontinuierliches Einlesen von einer Hauptkamera“, sowie den Aufruf der Methode „Einlesen der Nebenkameras“.

```
private void Class_Detection_ImageGrabbed(object sender, EventArgs e)
{
    //Grab finput Image from main camera (Top Camera)
    int i = 0; ; //Index for top camera
    Mat m = new Mat();
    capture[i].Retrieve(m);
    Input[i] = m.ToImage<Gray, byte>();

    Temp_Image[i] = MakeROIinImage(Input[i], rectangle[i]);
    Input[i] = m.ToImage<Gray, byte>();
    Temp_Input[i] = MakeBinaryImage(Temp_Image[i], capture_values[i, 3], capture_values[i, 2]);

    //Difference Image
    Difference[i] = GetDifferenceImage(Reference[i], Temp_Input[i]);

    //Get absolute value of the difference
    abs[i] = DartRecognition(Binary_Reference[i], Temp_Input[i]);

    //check if absolute value detects a dart on board
    if (abs[i] - 500 > temp_abs)
    {
        //Get intersection of two lines
        P_Intersection[i, throwed_darts] = GetIntersectionPoint(Difference[i],capture_values,i, throwed_darts);

        //Set new Threshold for absolute difference
        temp_abs = abs[i];
        detect = true;
        capture[i].Pause();

        //Grab from remaining cameras
        for (int j = 1; j < 4; j++)
        {
            GetDetectionfromallcameras(j);
            P_Intersection[j, throwed_darts] = GetIntersectionPoint(Difference[j], capture_values, j, throwed_darts);
        }
    }
}
```

Quelltext 13: Kontinuierliches Einlesen von der Hauptkamera und Differenzabgleich. Quelle: Eigene Darstellung.

Mit der Methode *GetDetectionfromallcameras()* werden die Bilder von den Nebenkameras eingelesen. In dieser Methode werden zusätzlich schon alle Verarbeitungen, z.B. BB- und Differenzbilderstellung, der Bilder durchgeführt. In dem Quelltext 13 werden schon die Schnittpunkte zwischen Symmetrielinie des Dartpfeils und Baseline errechnet. Diese Methode wird in Kapitel 6.2.8 dargestellt. In dem mehrdimensionalen Punktfeld *P\_Intersection[,j]* werden die Schnittpunkte von jeder Kamera zu jedem DP abgelegt.

## Priorisierung der Kamerabilder

```
private int Priorization(int [,] dartsize, int dartcount)
{
    int prio = 0;
    //Priolevel: 0...No prioritization possible
    //Priolevel: 2...Camtop and Camright
    //Priolevel: 3...Cambottom and Camright
    //Priolevel: 4...Camtop and Camleft
    //Priolevel: 5...Cambottom and Camleft

    //Check the dartsize of the horizontal cams
    if (dartsize[dartcount, 0] > dartsize[dartcount, 2])
        prio = 0;
    else
        prio = 1;
    //check the dartsize of the vertical cams
    if (dartsize[dartcount, 1] > dartsize[dartcount, 3])
        prio += 2;
    else
        prio += 4;

    return prio;
}
```

Quelltext 14: Methode zum Priorisieren des Kamerapaares,  
Quelle: Eigene Darstellung.

Wie in Kapitel 5.2.3 beschrieben, wird vor der Positionsauswertung eine Priorisierung durchgeführt. In dieser Priorisierung wird von den vier Kameras jeweils ein Kamerapaar, eine horizontale und eine vertikale Kamera ausgewählt, welche zur Positionsbestimmung des DP herangezogen wird. Die Priorisierung soll aus der Methode *GetIntersectionPoint()* die erfasste Größe des Rechteckes, welches den Schaft des DP umschließt, untereinander vergleichen. Das Umschließen des Schafts mit einem Rechteck wird im Quelltext 8 in Kapitel 6.2.7 beschrieben. Hier wird eine *variable area* deklariert, welche die Fläche des Rechtecks aus Breiten- und Längenmaß errechnet. Nach dem Vergleich gibt die

Methode *Priorization()* das qualitativ beste Kamerapaar zurück. In dem Quelltext 14 wird diese Methode beschrieben.

## Die Positionsbestimmung des DP auf der DS:

Wie auch die Detektion der DP wird auch die Positionsbestimmung in einer eigenen C#-Klasse erstellt. Um auf Methoden und Variablen einer Klasse zugreifen zu können, muss zuerst ein Objekt der zugreifende Klasse deklariert werden. Bsp.: *Class\_Positioning Positioning = new Class\_Positioning { }*. Nun kann mithilfe der Priorisierung die erforderlichen Parameter an die Methode *Scoring()* weitergegeben werden, welche die Punkte errechnet. Die Bestimmung der Punkte wird in Kapitel 7.2 erläutert. Der Zugriff auf die Methode wird im Quelltext 15 dargestellt. Die Methode gibt die errechnet Punkte als Ganzzahl zurück.

```
//Access to Scoring Class for Points evaluation
switch (temp_prio)
{
    case 0:
        points[thrown_darts] = 0;
        break;
    case 2:
        points[thrown_darts] = Positioning.Scoring(0, 1, P_Intersection[0, thrown_darts].X, P_Intersection[1, thrown_darts].X);
        break;
    case 3:
        points[thrown_darts] = Positioning.Scoring(2, 1, P_Intersection[0, thrown_darts].X, P_Intersection[1, thrown_darts].X);
        break;
    case 4:
        points[thrown_darts] = Positioning.Scoring(0, 1, P_Intersection[0, thrown_darts].X, P_Intersection[3, thrown_darts].X);
        break;
    case 5:
        points[thrown_darts] = Positioning.Scoring(2, 1, P_Intersection[0, thrown_darts].X, P_Intersection[3, thrown_darts].X);
        break;
    default:
        break;
}
```

Quelltext 15: Zugriff auf die Methode *Scoring()* der Klasse *Positioning{}*, Quelle: Eigene Darstellung.

**Probleme und Erkenntnisse:**

Das Hauptproblem des umgesetzten Algorithmus ist der Zugriff auf die Kameras während der Ausführung. Bei ungefähr 30% der Detektionsabwicklung stürzt das Programm aufgrund fehlender Referenzierung einer Kamera ab. Dies kommt daher, da durch die BUS-Last die *capture*-Objekte der Kameras ihre Referenzen verlieren. Da der Fehler nicht genau reproduzierbar ist, muss an einer Lösung des Problems weitergearbeitet werden.

In der Abbildung 59 wird dieser Fehler dargestellt.

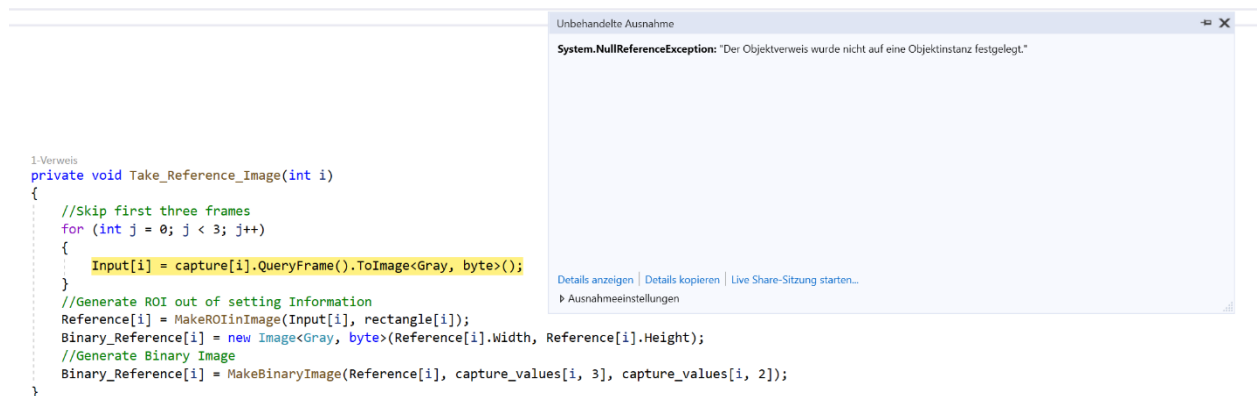


Abbildung 59: NullReferenceException des *capture()*-Objektes, Quelle: Eigene Darstellung.

Tritt der Fehler nicht ein, so führt der Algorithmus die geforderten Funktionen aus und kann zum Testen eingesetzt werden.

**7.3 Statistische Auswertung von erkannten DP**

Um eine statistische Auswertung der geworfenen DP zu erhalten, wird eine Testoberfläche für die Auswertung aufgesetzt. In der Abbildung 60 wird die Testumgebung dargestellt. Wie in Kapitel 5.2.3 beschreiben, kommt es beim simultanen Erfassen von vier Kameras zu häufigen Softwareabstürzen. Deswegen werden die statistischen Tests mit jeweils vordefinierten Kamerapaaren ausgeführt.

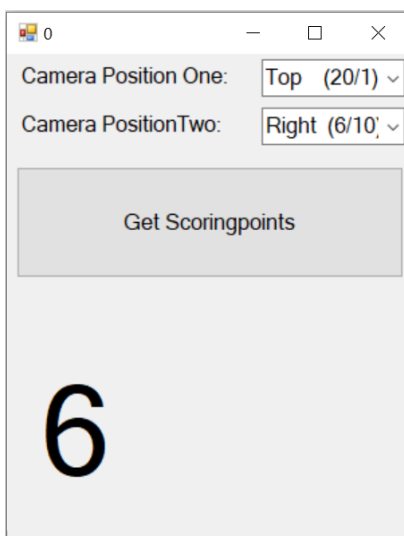


Abbildung 60: Testumgebung für die Statistische Auswertung, Quelle: Eigene Darstellung.

Diese Auswahl des Kamerapaars kann in der Testumgebung einfach in zwei Dropdown-Menüs durchgeführt werden. Wie in der Abbildung 60 beschrieben kann man im Dropdown Menü „Camera Positio One:“ die obere und untere Kamera und im Dropdown Menü:“ Camera Position Two“. Die jeweils rechte oder linke Kamera um die DS auswählen. Durch Drücken des Buttons „Get Scoring Points“ wird der Detektionsalgorithmus wie in Kapitel 7.2.4 beschrieben gestartet. Es werden bei einer Wurfserie jeweils drei DP ausgewertet und die erzielten Punkte pro DP in der Testoberfläche ausgegeben.

Um eine statistische Auswertung zu erhalten, ist eine Testreihe von 10 Wurfserien, 30 geworfene DP, pro Kamerapaar durchgeführt worden. In den Wurfserien sind die jeweiligen Parameter enthalten:

- Anzahl der DP: 30
- Anzahl der Doppelfelder (äußeres Segment des Sektors): 10
- Anzahl der Trippelfelder (mittleres Segment des Sektors): 10

Die Ergebnisse werden in der Tabelle 7 dargestellt.

Kamerapaar	Detektion des Darts	Sektorenerkennung	Doppelfeld Erkennung	Trippelfeld Erkennung
<b>Oben/Rechts</b>	28/30	20/30	5/10	6/10
<b>Oben/Links</b>	29/30	20/30	6/10	7/10
<b>Unten/Rechts</b>	25/30	22/30	6/10	6/10
<b>Unten/Links</b>	28/30	25/30	7/10	7/10
<b>Gesamt in [%]</b>	91,7	72,5	60	65

Abbildung 61: Auswertung der Wurfserien, Quelle: Eigene Darstellung.

Um die Ergebnisse besser auf das Gesamtsystem umzulegen, werden die Ergebnisse noch zusammengelegt. Dies bedeutet, dass, wenn ein Kamerapaar einen Pfeil nicht erkennt, dieser jedoch von einem anderen Kamerapaar erkannt wird, wird die Auswertung auf positive Erkennung angepasst. Die Testreihe wurde ident wiederholt und die DP sind jedes Mal gleich positioniert worden und mit allen Kamerapaaren getestet. Dadurch ergibt sich eine neue statistische Auswertung, welche in Tabelle 7 dargestellt wird.

	Detektion des Darts	Sektorenerkennung	Doppelfeld Erkennung	Trippelfeld Erkennung
<b>Gesamtsystem</b>	30/30	27/30	7/10	8/10
<b>Gesamt in [%]</b>	100	90	70	80

Abbildung 62: Statistik des Gesamtsystems, Quelle: Eigene Darstellung.

Durch die neuen Werte kann man herauslesen, dass die Probleme der Überlappung vollkommen ausgeschlossen und alle DP erkannt worden sind. Die Sektorenerkennung mit über 90% weist einen sehr guten Wert auf, der jedoch noch weiter gesteigert werden kann. Die gleiche Problematik gilt auch für Doppel- und Triplefeldererkennung. Diese Fehler können daher kommen, dass minimale Offsetfehler in der Positionserkennung entstehen. Diese treten auf, wenn die Kontur des Schafts, durch Spiegelungen am Schaft, nicht voll erfasst werden. In der Abbildung 63 wird dieser Fehler dargestellt. Die Symmetrieachse des DP wird am rechten äußeren Rand gezogen, dadurch wird die Achse leicht versetzt zur Baseline gezogen und der Schnittpunkt mit einem Offsetfehler errechnet

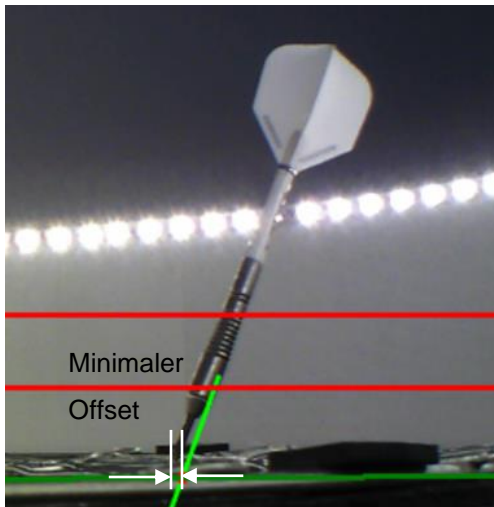


Abbildung 63: Offsetfehler durch schlechte Erkennung des Schaftes, Quelle: Eigene Darstellung.



## 8 FAZIT

Das Ziel dieser Masterarbeit lag darin, ein passendes Soft- und Hardwarekonzept zu erstellen und umzusetzen, welches ein automatisches Zählsystem für DP auf einer SDS durchführt. Durch eine Anforderungsanalyse von Hard- und Software wurde ein Softwarekonzept erstellt, welches die Umsetzung für den praktischen Teil darstellt. Nach Abschluss der Konzeptphase wurde ein praktischer Aufbau mit 3D-Druckteilen durchgeführt, der aus 3D-Druckteilen sowie Elektronikbauteilen aufgebaut wurde. In abschließenden Softwaretest wurden Ergebnisse ausgewertet und dargestellt.

### 8.1 Anforderungsanalyse und Konzepterstellung

In der Konzeptphase wurde eine Anforderungsanalyse durchgeführt, die die Anforderung an Software und Hardware darstellt. Durch klar definierte Anforderung konnte somit ein guter Überblick über Funktionen der Software aufgezeigt werden. Die Abläufe des zu erarbeiteten Algorithmus wurden dargestellt und mittels Flussdiagramme in eine grafische Form übermittelt.

Durch Umgebungsanalysen um eine DS konnte der geforderte Aufbau analysiert werden. Die dabei eruierten Faktoren, wie Lichteinfluss und baulichen Einschränkungen, wurden gesammelt und flossen in das Aufbaukonzept mit ein.

Die Zielsetzung der Anforderungsanalyse konnte somit klar definiert und die Ziele erfüllt werden.

### 8.2 Mechanischer Aufbau

Durch modulare 3D-Druckteile wurde der Aufbau durchgeführt. Die einzelnen Komponenten konnten so gestaltet werden, dass sie an jede Standarddartscheibe angebracht werden können. Die ausgewählten Kameras konnten gut in die Einhausung implementiert werden und sind dadurch auch vor abprallenden DP geschützt.

Durch Bereitstellen von Informationsblättern der angeschafften Komponenten ist es möglich, den Aufbau nachzubauen. Der mechanische Aufbau konnte vollständig durchgeführt werden und die Zielsetzung wurde damit erfüllt.

### 8.3 Erarbeitung des Softwarealgorithmus

Durch Aufsetzen einer Testumgebung in C# wurde eine grafische Oberfläche geschaffen, die es ermöglicht, Bilder von unterschiedlichen Kameras einzulesen und zu verarbeiten. Mithilfe von Einstellungsmöglichkeiten, welche für jede Kamera separat abgespeichert werden können, war es möglich, die Parameter für die Erkennung von DP durchzuführen. Mithilfe der Bildverarbeitungsbibliothek konnten die eingelesenen Bilder mit den Einstellparametern verarbeitet werden. In der Testumgebung konnte ein Algorithmus erarbeitet werden, der den DP erkennt und die Positionsbestimmung durchführen kann. Der Erkennungsalgorithmus konnte somit mit Mängeln durchgeführt werden. Diese Mängel zeigen sich in der Genauigkeit der Positionsfindung.

## 8.4 Erstellung der finalen Software und Softwaretest

Die Software konnte mithilfe der Informationen aus der Testumgebung erstellt werden. Der Such- und Positionierungsalgorithmus konnte umgesetzt werden. Die Probleme der BUS-Last und das damit verbundene Problem, dass nicht alle vier Kameras gleichzeitig verwendet werden können, stellt einen großen Mangel des Konzepts dar. Dadurch konnte die Software nicht vollständig getestet werden und muss vom Konzept her überarbeitet werden. Auch die finale Umsetzung auf einen EPC konnte nicht durchgeführt werden, da die Software noch nicht finalisiert wurde. Zusätzlich müssen in der Software noch Fehlerabfingroutinen eingebaut werden. Bei Eingabe von falschen Werten oder Setzen von inkorrekten Parametern ist die Software noch nicht vollständig gegen Fehler abgesichert.

Mit den gesammelten Informationen kann die Software jedoch verbessert werden. Die Software konnte somit nicht vollständig umgesetzt werden und kann auch noch nicht freigegeben und ausgerollt werden.

## 8.5 Ausblick

Das Grundkonzept, welches erarbeitet wurde, weist großes Potential zur optischen Auswertung von DP auf einer DS auf. Zu Beginn muss auf die Problematik der simultanen USB-BUS Nutzung eingegangen werden. Hierfür kann der Ansatz gewählt werden, die *VideoCapture()* der jeweils nicht genutzten Kameras zu entsorgen, um den Zugriff auf diese Kameras während des Ausführens des Algorithmus zu vermeiden. Sobald diese Problematik behoben worden ist, kann die Software auf einem EPC ausgerollt werden. Die Erstellung er unterschiedlichen Spielarten, sowie eine grafisch ansprechende Oberfläche muss erstellt werden. Durch Anbindung der Software an einen Webserver kann die Datenerfassung von unterschiedlichen Spielern erfasst werden. Onlinespiele können dadurch auch zusätzlich durchgeführt werden.

I

## LITERATURVERZEICHNIS

### Gedruckte Werke (8)

- Bradski, Gery; Kaebler, Adrian (2008): *Learning OpenCV*, 1 Auflage, O'Reilly Media Inc., San Francisco
- Burger, Wilhelm; Burge, Mark (2009): *Principles of Digital Image*, 1 Auflage, Springer Verlag London Ltd., London
- Efford, Nick (2000): *Digital Image Processing*, 2 Auflage, Licensing agency Ltd., Leeds
- Erhardt, Angelika (2008): *Einführung in die digitale Bildverarbeitung*, 1 Auflage, Vieweg und Teubner Fachverlag GmbH., Wiesbaden
- Gollapudi, Sunila (2019): *Learn Computer Vision Using OpenCV: With Deep Learning CNNs and RNNs*, 1 Auflage, Springer Verlag GmbH, Hyderabad, Telangana, India
- Hermes, Thorston (2005): *Digitale Bildverarbeitung*, 1 Auflage, Carl Hanser Verlag, München
- Kofler, Michale; Charly, Kühnast; Scherbeck, Christoph (2016): *Raspberry Pi-Das umfassende Handbuch*, 3 Auflage, Rheinwerk Verlag GmbH, Bonn
- Rozanski, Nick; Woods, Eoin (2012): *Software Systems Architecture*, 2 Auflage, Pearson Education Inc., Massachusetts

### Konferenzbeiträge (1)

- Qingcang, Yua; Chenga, Harry; Chengb, Wayne; Xiaodong, Zhou (2004): *Ch OpenCV for interactive open architecture computer vision*, in: Elsevier (Hrsg.): *Advances in Engineering Software*, SoftIntegration, Inc., Kalifornien, S. 10

### Online-Quellen (28)

- Scolia Technologies Ltd. (2020): *scoliadarts*  
<https://scoliadarts.com> [Stand: 21.05.2022]
- Bibliographisches Institut GmbH (2022): *Duden*  
<https://www.duden.de/rechtschreibung/Semantik> [Stand: 21.05.2022]
- Amazon (2022): *Amazon*  
<https://www.amazon.de/Roleo-Dart-Catchingring-Auffangring-Surround/dp/B016KDDPD4/> [Stand: 21.05.2022]
- Amazon (2022): *Amazon*  
[https://www.amazon.de/USB-Kameramodul-Weitwinkel-USB-Kameramodul-OV3660-Chip-2-0-Ausgang-Unterst%C3%BCtzung/dp/B088P1PKFM/ref=asc\\_df\\_B088P1PKFM/?tag=googshopde-21&linkCode=df0&hvadid=499385210395&hvpos=&hvnetw=g&hvrnd=1802306421006560202&hvpone=&hvptwo=&](https://www.amazon.de/USB-Kameramodul-Weitwinkel-USB-Kameramodul-OV3660-Chip-2-0-Ausgang-Unterst%C3%BCtzung/dp/B088P1PKFM/ref=asc_df_B088P1PKFM/?tag=googshopde-21&linkCode=df0&hvadid=499385210395&hvpos=&hvnetw=g&hvrnd=1802306421006560202&hvpone=&hvptwo=&) [Stand: 21.05.2022]
- Bjarki, Johannsson; Shruthi, Kashyap; Renukaprasad, Manjappa (2022): *delftswa*  
<https://delftswa.gitbooks.io/desosa2016/content/opencv/Chapter.html#opencv> [Stand: 21.05.2022]

Boldrow, Eugene (2020): *github.com*

<https://github.com/YellowFive5/OneHundredAndEighty> [Stand: 21.05.2022]

Brown, Eric (2021): *Linux Gizmos*

<https://linuxgizmos.com/150-open-spec-community-backed-linux-sbcs-under-200-3/> [Stand: 21.05.2022]

Canon (2022): *Canon*

<https://www.canon.com.au/get-inspired/glossary/fisheye-lens> [Stand: 21.05.2022]

Danner, Anika (2020): *isento*

<https://isento.de/blog/computer-vision/> [Stand: 21.05.2022]

Dimensions (2021): *Dimensions*

<https://www.dimensions.com/element/dartboard> [Stand: 21.05.2022]

Gimp (2010): *Documentation Gimp*

<https://docs.gimp.org/2.10/de/gimp-filter-desaturate.html#More-information-about-Luma> [Stand: 21.05.2022]

IBM (2022): *IBM*

<https://www.ibm.com/topics/computer-vision> [Stand: 21.05.2022]

Jörg, Brigitte; Uszkoreit, Hans (2015): *Universität Saarland*

[https://www.coli.uni-saarland.de/courses/is-is/slides/VLIWISKlassifikationen\\_Taxonomien.pdf](https://www.coli.uni-saarland.de/courses/is-is/slides/VLIWISKlassifikationen_Taxonomien.pdf) [Stand: 21.05.2022]

Krist, Alex (2021): *mydartpfeil.com*

<https://mydartpfeil.com/wdf->

[dartverband/#:~:text=Heutzutage%20stellt%20die%20WDF%20insgesamt,innerhalb%20eines%20Verbandes%20zu%20etablieren.](https://mydartpfeil.com/wdf-#:~:text=Heutzutage%20stellt%20die%20WDF%20insgesamt,innerhalb%20eines%20Verbandes%20zu%20etablieren.) [Stand: 21.05.2022]

Löbbering, Christian (2022): *pcwelt.de*

<http://www.pcwelt.de> [Stand: 21.05.2022]

Marr, Bernard (2019): *Forbes*

<https://www.forbes.com/sites/bernardmarr/2019/04/08/7-amazing-examples-of-computer-and-machine-vision-in-practice/?sh=4b00ff621018> [Stand: 21.05.2022]

MediaWiki (2022): *Emgu*

[https://www.emgu.com/wiki/index.php/Main\\_Page](https://www.emgu.com/wiki/index.php/Main_Page) [Stand: 21.05.2022]

Mulyana, Eueung (2016): *Slideshare*

<https://www.slideshare.net/e2m/single-board-computers-raspberry-pi-basics> [Stand: 21.05.2022]

MUO (2022): *makeuseof*

<https://www.makeuseof.com/tag/odroid-model-comparison/> [Stand: 21.05.2022]

Nelson, Joseph (2020): *Roboflow*

<https://blog.roboflow.com/yolov3-vs-mobilenet-vs-faster-rcnn> [Stand: 21.05.2022]

odroid (2017): *odroid.com*

<https://odroid.com/dokuwiki/doku.php#references> [Stand: 21.05.2022]

Online, Redaktion (2022): *LINUX Magazin Online*

<https://www.linux-magazin.de> [Stand: 21.05.2022]

OpenCV, Team (2022): *OpenCV*

<https://opencv.org/about/> [Stand: 21.05.2022]

Sablatnig, Robert; Zambanini, Sebastian; Licandro, Roxane (2014): *Technische Universität Wien*

[https://www.cg.tuwien.ac.at/courses/EinfVisComp/Skriptum/SS14/EVC-](https://www.cg.tuwien.ac.at/courses/EinfVisComp/Skriptum/SS14/EVC-02%20Einf%C3%BChrung%20in%20Computer%20Vision.pdf)

[02%20Einf%C3%BChrung%20in%20Computer%20Vision.pdf](https://www.cg.tuwien.ac.at/courses/EinfVisComp/Skriptum/SS14/EVC-02%20Einf%C3%BChrung%20in%20Computer%20Vision.pdf) [Stand: 21.05.2022]

Schnabel, Patrick (2022): *Elektronik Kombendium*

<https://www.elektronik-kompodium.de> [Stand: 21.05.2022]

Team, OpenCV (2022): *opencv.org*

<https://opencv.org/> [Stand: 21.05.2022]

wiki.odroid.com (2020): *ODROID Wiki*

<https://wiki.odroid.com> [Stand: 21.05.2022]

YellowFive5 (2020): *GitHub*

<https://github.com/YellowFive5/DartboardRecognition> [Stand: 21.05.2022]

#### **Wissenschaftliche Artikel (4)**

Weleba, Dominikus (Hrsg.) (2020): *Auswertung von Steeldartpfeilen: Optische Auswertung von Steeldartpfeilen auf einer Dartscheibe*

Kaler, Promad (Hrsg.) (2010): *Grayscale Image: Study of Grayscale image in Image processing*

De Vries, Elmer (Hrsg.) (2004): *Object Recognition: Object Recognition: A Shape-Based Approach using Artificial Neural Networks*

Xhensila, Poda (Hrsg.) (2010): *Shape detection: Shape detection and classification using OpenCV and Arduino Uno*

## ABBILDUNGSVERZEICHNIS

Abbildung 1: Aufbau Scolia System, Quelle: Scolia Technologies Ltd. (2020), Online-Quelle [21.05.2022]. .....	2
Abbildung 2: OHEC System, schematischer Aufbau, Quelle: YellowFive5 (2020), Online-Quelle [21.05.2022]......	3
Abbildung 3: Feature-Matrix Tabelle von EPC, Quelle: Brown (2021), Online-Quelle [21.05.2022]. ....	5
Abbildung 4: Preis-Leistung-Darstellung von EPC, Quelle: Brown (2021), Online-Quelle [21.05.2022]. ....	6
Abbildung 5: RP 3 B Hardwareaufbau, Quelle: Kofler/Charly/Scherbeck (2016) (leicht modifiziert), S. 23. .....	10
Abbildung 6: OD C4 Hardwareaufbau, Quelle: wiki.ordoid.com (2020), Online-Quelle [21.05.2022]......	13
Abbildung 7: Probleme der CV, Quelle: Nelson (2020), Online-Quelle [21.05.2022]. ....	15
Abbildung 8: Montierte Kamera auf einem Rückspiegel: Quelle: Bradski/Kaebler (2008), S. 4 (leicht modifiziert). ....	16
Abbildung 9: Darstellung des Eiffelturms von mehreren Perspektiven, Quelle: Gollapudi (2019), S. 26. ...	17
Abbildung 10: Beispiel der affinen Transformation von Objekten, Quelle: Object Recognition (2004), S. 8 (leicht modifiziert). ....	18
Abbildung 11: RGB-Farbwürfel, Quelle: Efford (2000), S. 28. ....	19
Abbildung 12: Binärbild aus einem Grauwertbild, Quelle: Hermes (2005), S. 30. ....	21
Abbildung 13: Blurring eines Bildes, Quelle: Burger/Burge (2009), S. 98.....	22
Abbildung 14: Filtermatrix und Hot Spot, Quelle: Burger/Burge (2009). S. 100.....	23
Abbildung 15: Anwendung einer Filter-Matrix, Quelle: Burger/Burge (2009), S. 101. ....	24
Abbildung 16: Grenzen von Filtern, Quelle: Burger/Burge (2009), S. 107. ....	24
Abbildung 17: Ideale Kante anhand eines Grauwertes, Quelle: Hermes (2005) (leicht modifiziert), S. 70. .....	25
Abbildung 18: Übersicht OPC, Quelle: Bjarki/Shruthi/Renukaprasad (2022), Online-Quelle [21.05.2022]. .....	29
Abbildung 19: Übersicht der benutzen Programmiersprachen für OCV, Quelle: Bjarki/Shruthi/Renukaprasad (2022), Online-Quelle [21.05.2022]. ....	30
Abbildung 20: Basisstruktur von OCV, Quelle: Bradski/Kaebler (2008), S. 13. ....	31
Abbildung 21: Positionierung eines Vierkameranysystem um die DS, Quelle: Eigene Darstellung. ....	34
Abbildung 22: Fischaugeneffekt einer Kamera mit mehr als 110° Erfassungswinkel, Quelle: Auswertung von Steeldartpfeilen (2020), S. 34 (leicht modifiziert). ....	34

Abbildung 23: Überlappung von zwei Dartpfeilen, Quelle: Auswertung von Steeldartpfeilen (2020), S. 20.  
..... 35

Abbildung 24: Dartring zum Schutz der dahinterliegenden Mauer, Quelle: Amazon (2022), Online-Quelle  
[21.05.2022]. ..... 36

Abbildung 25: Entfernung eines DP zu einer Kamera, Quelle: Eigene Darstellung. .... 38

Abbildung 26: Schritte zur Findung der Symmetrieachse eines DP, Quelle: Eigene Darstellung. .... 38

Abbildung 27: Gesamtkonzept für den Hardwareaufbau, Quelle: Eigene Darstellung. .... 39

Abbildung 28: Festlegen der Parameter- und Sucheinstellungen, Quelle: Eigene Darstellung..... 41

Abbildung 29: Ablauf der Erkennung eines DP, Quelle: Eigene Darstellung. .... 42

Abbildung 30: Veränderung auf einer DS, Quelle: Eigene Darstellung. .... 43

Abbildung 31: Koordinatenbestimmung mittels Reduktion auf Symmetrielinie des DP, Quelle: Eigene  
Darstellung. .... 44

Abbildung 32: Hardwareaufbau der DS, Quelle: Eigene Darstellung. .... 45

Abbildung 33: Einzelmodule für die Einhausung, Quelle: Eigene Darstellung..... 46

Abbildung 34: Klemmbracken, Quelle: Eigene Darstellung. .... 46

Abbildung 35: 3D-Druck eines kompletten Drucksets der Einhausung, Quelle: Eigene Darstellung. .... 47

Abbildung 36: LED-Klebestreifen und Durchbruch in der Einhausung, Quelle: Eigene Darstellung. .... 47

Abbildung 37: USB-Verteiler, Quelle: Eigene Darstellung. .... 48

Abbildung 38: Kameramodule, Quelle: Amazon (2022), Online-Quelle [16.Mai.2022]. ..... 48

Abbildung 39: GUI der Softwaretestumgebung, Quelle: Eigene Darstellung..... 49

Abbildung 40: Einbinden von ECV in ein C# Projekt, Quelle: Eigene Darstellung. .... 50

Abbildung 41: Einlesen des Originalbildes einer Kamer, Quelle: Eigene Darstellung. .... 51

Abbildung 42: Kamera- und Auflösungsauswahl, Quelle: Eigene Darstellung. .... 51

Abbildung 43: Umwandeln des Farbbildes in ein GSB, Quelle: Eigene Darstellung. .... 52

Abbildung 44: BB mit unterschiedlichen Schwellwerten, Quelle: Eigene Darstellung. .... 53

Abbildung 45: Auswahl der unterschiedlichen Filterarten, Quelle: Eigene Darstellung. .... 54

Abbildung 46: Unterschiede der Filtertypen, Quelle: Eigne Darstellung. .... 55

Abbildung 47: Darstellung eines Differenzbildes vor und nach einem DP-Wurfes, Quelle: Eigene  
Darstellung. .... 55

Abbildung 48: Positionierung und Größenänderung der ROI, Quelle: Eigene Darstellung. .... 56

Abbildung 49: Umschließen von drei DP mit Rechtecken, Quelle: Eigene Darstellung. .... 57

Abbildung 50: Schnittpunktbestimmung der DS und der Symmetrieachse von DP, Quelle: Eigene Darstellung. ....	58
Abbildung 51: Fehlstellung der Symmetrielinie des DP, Quelle: Eigene Darstellung. ....	59
Abbildung 52: Aufrufen der Settings in C#, Quelle: Eigene Darstellung. ....	60
Abbildung 53: Beispiel für Erstellen der Kamer ID Parameter, Quelle: Eigene Darstellung. ....	60
Abbildung 54: Beispiel zum Einstellen der Kamerakalibrierung für die linke Kamera, Quelle: Eigene Darstellung. ....	61
Abbildung 55: Trichterförmiges Erfassen einer Kamera, Quelle Eigene Darstellung. ....	64
Abbildung 56: Schrittweise Erläuterung der Schnittpunktberechnung, Quelle: Eigene Darstellung. ....	65
Abbildung 57: Abmessungen eines Dartboards, Quelle: Dimensions (2021), Online-Quelle [21.05.2022] (leicht modifiziert). ....	67
Abbildung 58: Ablauf des Detektionsalgorithmus, Quelle: Eigene Darstellung.....	68
Abbildung 59: NullReferenceException des <i>capture()</i> -Objektes, Quelle: Eigene Darstellung.....	72
Abbildung 60: Testumgebung für die Statistische Auswertung, Quelle: Eigene Darstellung.....	72
Abbildung 61: Auswertung der Wurfserien, Quelle: Eigene Darstellung.....	73
Abbildung 62: Statistik des Gesamtsystems, Quelle: Eigene Darstellung. ....	73
Abbildung 63: Offsetfehler durch schlechte Erkennung des Schaftes, Quelle: Eigene Darstellung. ....	74



## **TABELLENVERZEICHNIS**

Tabelle 1: Informationen SCS, Quelle: Eigenen Darstellung. ....	2
Tabelle 2: Informationen OHEC, Quelle: Eigene Darstellung. ....	3
Tabelle 3: Vergleich der Raspberry Pi-Modelle, Quelle: Eigene Darstellung.....	9
Tabelle 4: Vergleich der Odroid-Modelle, Quelle: Eigene Darstellung.....	12
Tabelle 5: Stakeholder von OCV, Quelle: Eigene Darstellung. ....	28
Tabelle 6: Beschreibung der Einstellparameter, Quelle: Eigene Darstellung. ....	62
Tabelle 7: Fortsetzung von Tabelle 6, Quelle: Eigene Darstellung. ....	63

## QUELLTEXTVERZEICHNIS

Quelltext 1: Einbindung von ECV in eine C#-Klasse, Quelle: Eigene Darstellung.....	50
Quelltext 2: Quelltextausschnitt zum Einlesen aus von einer Kamer bis zur grafischen Ausgabe, Quelle: Eigene Darstellung. ....	51
Quelltext 3: Funktion zum Auslesen der Kamerahardware-ID mittels Kameraindex, Quelle: Eigene Darstellung. ....	52
Quelltext 4: Methode zum Auslesen des Kameraindex mittels Kamerahardware-ID, Quelle: Eigene Darstellung. ....	52
Quelltext 5: Schwellwertereinstellungen für das BB mittels Schieberegler, Quelle: Eigene Darstellung. ....	53
Quelltext 6: Erstellung der Bilder mit den Filtereinstellungen, Quelle: Eigene Darstellung. ....	54
Quelltext 7: Erstellen einer ROI, Quelle: Eigene Darstellung. ....	56
Quelltext 8: Quelltextausschnitt zur Findung von gedrehten Rechtecken, Quelle: Eigene Darstellung. ...	57
Quelltext 9: Bestimmung des Schnittpunktes zweier Geraden, Quelle: Eigene Darstellung. ....	59
Quelltext 10: Abfrage, in welchen Quadranten sich der DP befindet, und Ziehen der Linie zur Berechnung des Schnittpunktes, Quelle: Eigene Darstellung. ....	66
Quelltext 11: Erstellung der <i>VideoCapture()</i> Objekte mit den jeweiligen Kameraeinstellungen, Quelle: Eigene Darstellung. ....	69
Quelltext 12: Referenzbild einlesen für alle Kameras, Quelle: Eigene Darstellung. ....	69
Quelltext 13: Kontinuierliches Einlesen von der Hauptkamera und Differenzabgleich, Quelle: Eigene Darstellung. ....	70
Quelltext 14: Methode zum Priorisieren des Kamerapaars, Quelle: Eigene Darstellung. ....	71
Quelltext 15: Zugriff auf die Methode <i>Scoring()</i> der Klasse <i>Positioning{}</i> , Quelle: Eigene Darstellung. ....	71

## **ABKÜRZUNGSVERZEICHNIS**

BB	Binärbild
BS	Betriebssystem
CV	Computer Vision
DS	Dartscheibe
ECV	EmguCV
EPC	Einplatinencomputer
GSB	Graustufenbild
GUI	Graphical user Interface
OCV	OpenCV
OD	Odroid
OHEC	OneHundredAndEightyCore
RP	Raspberry Pi
SCS	Scolia System
SD	Stelldart
SWT	Softwaretestumgebung