**Institut für Montagetechnik**

Prof. Dr.-Ing. A. Raatz

Studien-/Diplom/-Bachelor-/Masterarbeit

„Verkleinern des Real2Sim-Gaps einer Präzisionsmontageanlage durch die automatische Vermessung der Roboterperipherie“

Verfasser:

Yue Ju

Matrikel-Nr.: 10043050

Betreuer:

Rolf Wiemann

Garbsen, den 16.08.2023

**Thema der Arbeit:** Studienarbeit

**Erstellt von:** Yue Ju

Matrikelnummer 10043050

**Kenn-Nr.:** 03/23

**Abgabedatum:** 16.08.2023

**Betreuer:** Rolf Wiemann

Erklärung

Hiermit versichere ich, dass ich die vorliegende Studienarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Passagen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, sind als solche kenntlich gemacht. Ich versichere darüber hinaus, dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorliegt.

Hannover, den 16.08.2023 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Yue Ju

**Abstract**

In this work, the Real2Sim gap of the vacuum gripper of a precision assembly system is reduced. For this purpose, a robot operating system ROS 2 is mainly used in the work to control the gripper and detection of the gripper is carried out with the OpenCV program library. The simulation model is automatically calibrated by an adaptation of the specific file at last. The accuracy achieved is approximately 97.7305%.

# Inhaltverzeichnis

[Inhaltverzeichnis VII](#_Toc142566603)

[Abbildungsverzeichnis VIII](#_Toc142566604)

[Abkürzungsverzeichnis: X](#_Toc142566605)

[1. Einleitung 1](#_Toc142566606)

[2. Theorieteil 2](#_Toc142566607)

[2.1 Arbeitsaufbau 2](#_Toc142566608)

[2.2 Real2sim-Gap 3](#_Toc142566609)

[2.3 Das Roboter-Betriebssystem ROS 2 4](#_Toc142566610)

[2.3 Detektion mit OpenCV 7](#_Toc142566611)

[2.3.1 Canny-Algorithmus 7](#_Toc142566612)

[2.3.2 Konturenerfassung 14](#_Toc142566613)

[2.3.3 Kreisdetektion 17](#_Toc142566614)

[3. Arbeitsdurchführung 19](#_Toc142566615)

[3.1 Kalibrierungsvorgehensweise 19](#_Toc142566616)

[3.2 Hauptlogik des Programms 20](#_Toc142566617)

[3.3 Steuern der Trajektorie 21](#_Toc142566618)

[3.3.1 Steuerungsmethode 21](#_Toc142566619)

[3.3.2 Programmierteil 25](#_Toc142566620)

[3.3.3 Rotationstrajektorie 25](#_Toc142566621)

[3.4 Detektion des Greifers 26](#_Toc142566622)

[3.4.1 Rauschminderung mit Median-filter 26](#_Toc142566623)

[3.4.2 Kantendetektion mit Canny-Algorithmus 33](#_Toc142566624)

[3.4.3 Kreisdetektionsmethode 34](#_Toc142566625)

[3.4.4 Programmsdurchführung 41](#_Toc142566626)

[3.5 Genauigkeitsanalyse und Fehleranalyse 42](#_Toc142566627)

[3.5.1 genauigkeit 42](#_Toc142566628)

[3.5.2 Fehleranalyse 44](#_Toc142566629)

[3.6 Kalibrierung des Modells 44](#_Toc142566630)

[4. Zusammenfassng und Ausblick 46](#_Toc142566631)

[5. Literaturverzeichnis 48](#_Toc142566632)

# Abbildungsverzeichnis

[Abbildung 1: Die Präzisionsmontageanlage 3](#_Toc142565304)

[Abbildung 2: Das ROS-Ökosystem 4](#_Toc142565305)

[Abbildung 3: Die Pixelintensitätsänderung der Kante 8](#_Toc142565306)

[Abbildung 4: Der Pixelintensitätstrend eines 1D-Bildes 8](#_Toc142565307)

[Abbildung 5: Die erste Ableitung der Intensitätsänderung 9](#_Toc142565308)

[Abbildung 6: Die Faltung in einem 2D-Bild 9](#_Toc142565309)

[Abbildung 7: Die zweite Ableitung der Intensitätsänderung 10](#_Toc142565310)

[Abbildung 8: Die Richtungen des Gradienten 11](#_Toc142565311)

[Abbildung 9: Unterdrückung nicht maximales Werts 12](#_Toc142565312)

[Abbildung 10: Hysterese-Schwellenwert 13](#_Toc142565313)

[Abbildung 11: Detektierte Konturpunkte mit CHAIN\_APPROX\_NONE(links) und CHAIN\_APPROX\_SIMPLE(rechts) 16](#_Toc142565314)

[Abbildung 12: Das Greifersmodell in der Simulation 17](#_Toc142565315)

[Abbildung 13: Das mit der Kamera aufgenommene Greifersbild 18](#_Toc142565316)

[Abbildung 14: Die Bewegungsspur der Mitte des Greifers 20](#_Toc142565317)

[Abbildung 15: Die „Action“-Kommunikation 23](#_Toc142565318)

[Abbildung 16: Die Lücke an der unteren Ebene 27](#_Toc142565319)

[Abbildung 17: Die Lücke unter dunkler Beleuchtung 28](#_Toc142565320)

[*Abbildung 18: Das Detektionsergebnis* 29](#_Toc142565321)

[Abbildung 19: Die Funtionsweise eines 3 × 3 Median-Filters 30](#_Toc142565322)

[Abbildung 20: Detektion ohne Bildverarbeitung 31](#_Toc142565323)

[Abbildung 21: Detektion mit der Blendegröße 3, 5, 7, 9 von Median-Filter(von oben links nach rechts) 32](#_Toc142565324)

[Abbildung 22: Zweidimensionale Normalverteilung 33](#_Toc142565325)

[Abbildung 23: Detektion mit der Blendegröße 3, 5, 7, 9 von gaußschem Filter(von oben links nach rechts) 33](#_Toc142565326)

[Abbildung 24: Canny-Detektion mit „apertureSize“ 3, 5, 7 (von oben links nach rechts) 35](#_Toc142565327)

[Abbildung 25: Interpolation mit fitEllipse() der drei Versuche (von oben links nach rechts) 39](#_Toc142565328)

[Abbildung 26: Interpolation mit fitEllipseAMS() der drei Versuche (von oben links nach rechts) 40](#_Toc142565329)

[Abbildung 27: Interpolation mit fitEllipseDirect() der drei Versuche (von oben links nach rechts) 40](#_Toc142565330)

[Abbildung 28: Interpolation mit hyperSVD() (Versuch 1) 41](#_Toc142565331)

[Abbildung 29: Interpolation mit hyperSVD() (Versuch 2) 42](#_Toc142565332)

[Abbildung 30: Interpolation mit hyperSVD() (Versuch 3) 42](#_Toc142565333)

[Abbildung 31: Kalibeirungsergebnisse der drei Versuche mit Abweichung in y-Achse (von oben links nach rechts) 44](#_Toc142565334)

[Abbildung 32: Kalibeirungsergebnisse der drei Versuche mit Abweichung in x-Achse und y-Achse (von oben links nach rechts) 45](#_Toc142565335)

# Abkürzungsverzeichnis:

bzw.: beziehungsweise

ca.: circa.

d. h.: das heißt

DDS: Data Distribution Service

IDL: Interface Definition Language

ROI: region of interest

z. B. zum Beispiel

µm: Mikrometer

# 1. Einleitung

In der Präzisionsmontage ist die Entwicklung für neue Varianten oder Produkte immer aufwändig. Der Grund liegt daran, dass das Merkmal der Präzisionsmontage eine hohe Variantenvielfalt bei geringen Stückzahlen ist und dabei werden die Qualität und Genauigkeit ebenfalls anspruchsvoll angefordert. Das führt zu einem sehr hohen Programmieraufwand neben der Entwicklung der entsprechenden Hardware für die Montageprozesse.

Um die daraus resultierenden hohen Stillstandzeiten der Maschine zu vermeiden, wird der gesamte Montageprozess in einer Simulationsumgebung ausgeführt. Dazu wurde bereits ein Simulationsmodell der Match-Präzisionsmontagezelle in Gazebo implementiert. Aber wenn ein Montageprozess in der Simulationsumgebung durchgeführt wird, muss das Simulationsmodell auf das reale System angepasst. Andernfalls hat es eine Abweichung zwischen der Simulation und realität, bzw. einen Real2Sim-Gap zur Folge.

Ziel dieser Arbeit ist die Verringerung dieses Real2Sim-Gaps eines Vakuumgreifers durch die automatische Kalibrierung der Simulation. Aus technischen Gründen wird der Greifer durch die Schrauben montiert, was zu eine kleine Abweichung der Position des Greifers führt. Da das aktuelle Simulationsmodell der Präzisionsmontageanlage auf dem CAD-Modell basiert, stimmen die Pose des Greifers nicht mit den realen Posen überein. Es wird daher angefordert, dass die Position des Greifers in dem Modell kalibriert wird, um die Simulation bestmöglich der Realität anzupassen. Die Kalibrierung wird auf der Basis von OpenCV durchgeführt. Außerdem wird die Steuerung des Roboters mit der Hilfe von ROS erreicht. Zur Vermessung des… wird auch die Unterseitenkamera der Präzisionsmontageanlage eingesetzt. Und Schließlich wird die Postion des Greifers in der Simulation angepasst, sodass ein möglichst genaues Simulationsmodell der Realität entspricht.

In dieser Arbeit werden zuerst die Vorbereitungen über das Hardware und Software und die relevante Theorie erläutert. Anschließend werden die zwei Hauptteile erklärt, nämlich die Steuerung des Roboters und die Bild-Detektion mit OpenCV. Dann erfolgt die Anpassung der bestimmten Datei. Schließlich werden die erarbeitete Lösung und einige mögliche Verbesserungen diskutiert.

# 2. Theorieteil

In diesem Kapitel werden hauptsätzlich der Arbeitsaufbau und das erforderliche theoretische Wissen erklärt.

## 2.1 Arbeitsaufbau

Die Durchführung dieser Arbeit basiert auf einer Präzisionsmontageanlage, siehe Abbildung 1. Der zu kalibrierte Bauteil ist der Vakuumgreifer „SPT\_Vacuumtool“. Davon relevant für die Kalibrieraufgabe sind die Unterseitenkamera, der Spiegel sowie die vier Bewegungsachsen, nämlich die X-, Y-, Z-Achse und eine drehbare rote Platte, die die Z-Achse und den Greifer verbindet. Die Unterseitenkamera und der Greifer sind vertikal zueinander ausgerichtet. Der Spiegel befindet sich am Schnittpunkt ihrer senkrechten Linien und ist im 45° angebracht. Das in der Unterseitenkamera verwendete Objektiv ist ein telezentrisches Objektiv. Telezentrische Objektive vermeiden den Parallaxenfehler, die bei herkömmlichen Objektiven auftreten. Außerdem weist telezentrische Objektive eine sehr geringe Verzeichnung auf, was die Genauigkeit der Vermessung weiter verbessert [1]. Der Sensor der Kamera besitzt eine pixelgröße von 2.2 µm × 2.2 µm. Beim Fotografieren wird eine Ring-beleuchtung benutzt. Sie beleuchtet durch einen Ring entlang des Objektivs. Die Beleuchtungsstärke beträgt 40%.

Die verwendete Programmiersprache ist Python(Version: 3.10.6). Die Programmierumgebung ist Visual Studio Code. Das Betriebssystem ist Linux.

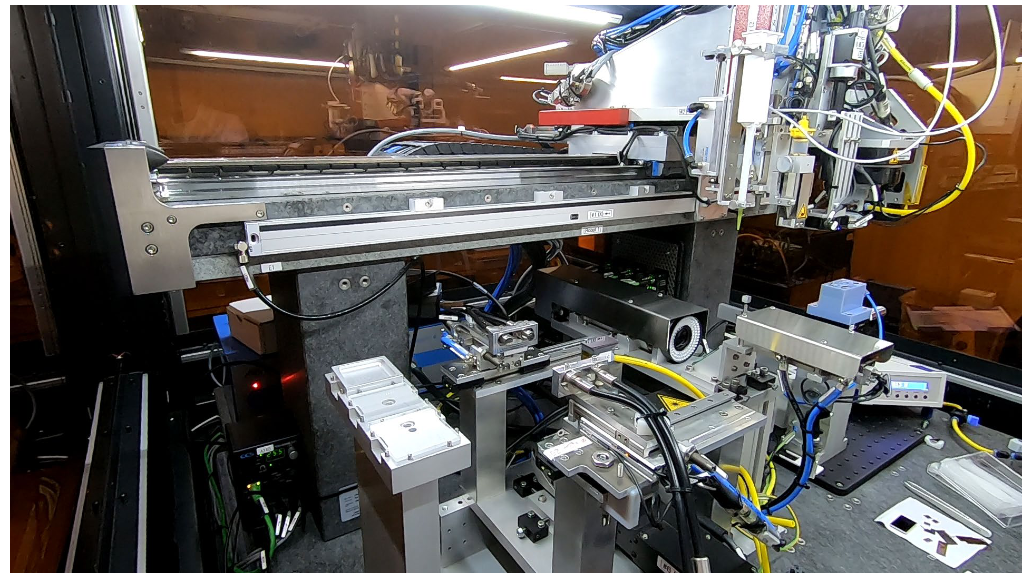


Abbildung 1: Die Präzisionsmontageanlage[2]

## Real2Sim-Gap

Heutzutage werden viele Entwicklungsaufgaben in der Simulationsumgebung durchgeführt. Die Verwendung von Simulationsumgebungen senkt nicht nur die Entwicklungskosten, sondern verbessert auch die Test- und Entwicklungszyklen und bietet die Entwicklung eine hohe Flexibilität. Aber eine Simulation, die nicht gut mit der Realität übereinstimmt, führt zu dem Mißerfolg der Aufgabe. Der Grund dafür ist die Existenz der lücken zwischen der Realität und Simulation, bzw. der Real2Sim-Gap oder der Sim2Real-Gap. Der Gap wird verursacht durch z. B. das Licht, die Vibration usw. in der Realität [3].

In dieser Arbeit wird der Real2Sim-Gap, die durch die Verschiebungen des Objektes verursacht, betrachtet. Der Vakuumgreifer ist mit den Schrauben an die rote Platte befestigt, wobei eine radiale Verschiebung zwischen dem Greifer und der Platte entsteht. Das Simulationsmodell werden ideal gebaut und deswegen gibt es eine Abeichung zwischen der Realität und Simulation beim Anschluss dieser zwei Teile.

## 2.3 Das Roboter-Betriebssystem ROS 2

Das Robot Operating System, abgekürzt mit ROS, ist ein Framework für persönliche Roboter und Industrieroboter. ROS ist kein Betriebssystem. Vielmehr handelt es sich um ein SDK (Software Development Kit), das die Bausteine bereitstellt, die zum Erstellen der Roboteranwendungen benötigt sind. Diese Arbeit wird mit ROS als den Rahmen durchgeführt [4].



Abbildung 2: Das ROS-Ökosystem[4]

Wie die Abbildung 2 gezeigt, besteht das ROS-Ökosystem aus vier Teilen: „plumbing“, „tools“, „capabilities“ und „community“. Im Kern stellt ROS ein Nachrichtenübermittlungssystem bereit, das oft als „Middleware“ oder „Plumbing“ bezeichnet wird. Dies Nachrichtensystem verwaltet die Details der Kommunikation zwischen „Node“. Ein Node ist ein Prozess, der Berechnungen durchführt. Nodes werden zu einem Diagramm zusammengefasst und kommunizieren über ein anonymes Publish/Subscribe-Modell [5]. Bei der Entwicklung von Roboteranwendungen gibt es viele Schwierigkeiten und die „tools“ in ROS helfen gut dabei. ROS hat viele Werkzeuge einschließlich: Start, Selbstbeobachtung, Debugging, Visualisierung, Plotten und so weier. Außerdem bietet ROS die Bausteine von Treibern über Algorithmen bis hin zu Benutzeroberflächen, was die „capabilities“ von ROS verkörpert. Die „community“ von ROS ist groß, vielfältig und global. Ingenieure und Hobbyisten aus den ganzen Welt halten das ROS-Projekt am Laufen.

In dieser Arbeit wird die zweite Generation des Roboter-Betriebssystems(ROS 2) benutzt. ROS 1 wurde vom Robotik-Inkubator Willow Garage popularisiert. ROS 1 hat in fast allen Bereichen intelligenter Maschinen Wirkung gezeigt. Als die kommerzielle Möglichkeiten in die Produkte übergegangen, zeigten die Grundlagen von der ersten Generation des Roboter-Betriebssystems(ROS 1) als Forschungsplattform ihre Grenzen. In ROS 1 haben die Sicherheit, Netzwerktopologie und Systemverfügbarkeit keine Priorität. Aber mit der Entwicklung der Roboterindustrie werden die Sicherheit, Zuverlässigkeit in nicht-traditionellen Umgebungen und die Unterstützung großer eingebetteter Systeme von immer größer Bedeutung. Durch eine komplett neue Gestaltung, meistern ROS 2 diese Herausforderungen gut. ROS 2 basiert auf dem Data Distribution Service (DDS). Dieser ist ein offener Kommunikationsstandard, der in kritischen Infrastrukturen wie Militär, Raumfahrzeugen und Finanzsystemen verwendet wird. Es löst viele Probleme beim Aufbau zuverlässiger Robotersysteme. DDS ermöglicht ROS 2 erstklassige Sicherheit, eingebettete und Echtzeitunterstützung, Multi-Roboter-Kommunikation und Betrieb in nicht idealen Netzwerkumgebungen [6].

Um das Steuern des Roboters mit ROS zu erreichen, muss die grundlegende Kommunikation zwischen „Nodes“ ausgeführt. Ein „Node“, das vorher schon erwähnt wurde, ist für einen einzigen, modularen Zweck in ROS verantwortlich [7]. Ein vollständiges Robotersystem besteht aus vielen Nodes, die zusammenarbeiten. Jedes Node kann die Daten senden oder von anderen Noden die Daten empfangen, wodurch eine Kommunikation zwischen „Nodes“ erreicht wird.

ROS-Anwendungen kommunizieren typischerweise über unterschiedliche Typen der Kommunikationsmodi in ROS. In ROS 2 gibt es insgesamt drei Typen der Kommunikationsmodi zwischen Nodes: Topic, Service und Action. Diese Kommunikationsmodi basieren auf drei unterstützten Basisschnittstellen.

ROS 2 verwendet eine vereinfachte Beschreibungssprache, die Interface Definition Language (IDL), um diese Schnittstellen zu beschreiben. Diese Beschreibung erleichtert die ROS-Werkzeuge, den Quellcode für den Schnittstellentyp in mehreren Zielsprachen automatisch zu generieren. Die drei Schnittstellen sind: „msg“, „service“, „action“, welche bedeuten Message, Service und Aktion [8].

„msg“: Eine Message in ROS ist verwendet in der Topic-Kommunikation und ist beschreibt durch die „.msg“-Datei, die aus einfache Texte besteht. Die „.msg“-Datei wird verwendet, um den Quellcode für Message in verschiedenen Programmiersprachen zu generieren.

„service“: Ein Service entspricht der Service-Komunnikation und ist beschreibt durch „.srv“-Datei, die aus einer Anfrage und einer Antwort besteht. Die Anfrage und die Antwort sind jeweils eine Message.

„action“: Eine Aktion entspricht der Action-Kommunikation. Sie werden durch „.action“-Dateien beschreiben und bestehen aus drei Teilen: einem Ziel, einem Ergebnis und Feedback. Jeder Teil ist selbst eine Message.

„Topic“ ist eine Hauptmethode, mit denen die Daten zwischen den Nodes übertragen werden. Sie verwendet ein “Publisher-Subscriber”-Modell um einen Datenfluss zu erreichen. Die Nodes mit gleichen Namen der “Topics” kommunizieren miteinander. Ein Node kann die Daten zu einer beliebigen Anzahl von “Topics” publizieren und auch eine beliebige Anzahl von “Topics” abonnieren. „Service“ basiert auf einem „Call-and-Response“-Modell. Anders als einen kontinuierlichen Datenfluss von der „Topic“-Kommunikation, stellt die „Service“-Kommunikation die Daten nur dann bereit, wenn sie ausdrücklich von einem Client aufgerufen werden. Es kann viele Clients existieren , die einen selben Service benutzten, aber für jeder Service kann es nur ein einzelner Server existieren [6]. „Action“ ist für lang laufende Aufgaben gedacht. Sie bestehen aus drei Teilen: einem Ziel, Feedbacks und einem Ergebnis. Die „Action“-Kommunikation benutzt ein „Client-Server“-Modell, das auf „Topic“ und „Service“ basiert. Ein „Action“-Client sendet ein Ziel an einen Server, der das Ziel bestätigt und einen Feedbackstrom und ein Ergebnis zurückgibt. Eine Haupteigenschaft der „Action“-Kommunikation ist, dass sie abgebrochen während der Ausführung werden kann. Außerdem kann die „Action“-Kommunikation stetige Feedbacks während der Ausführung neben der einzelnen endlichen Antwort des Ergebnisses geben.

## Detektion mit OpenCV

OpenCV ist eine freie Programmbibliothek mit Algorithmen für die Bildverarbeitung und Computer Vision. Sie besitzt mehrere Hundert Computer Vision-Algorithmen [9]. Die Entwicklung der Bibliothek wurde von Intel initiiert und bis 2013 von Willow Garage gepflegt und nachher wieder von Intel übernommen wurde. OpenCV besitzt eine schnelle Geschwindigkeit und eine große Menge der Algorithmen aus neuesten Forschungsergebnissen [10]. Außerdem werden die Algorithmen in die entsprechenden Funktionen gekapselt und deswegen ist es ziemlich bequem um die bestimmten Aufgaben zu erledigen mit OpenCV.

In dieser Arbeit wird OpenCV hauptsätzlich benutzt zur Detektion des Vakuumgreifers. Dafür wird es zuerst die Verarbeitung zu den aufgenommenen Bilder implementiert. Wegen einer kreisförmigen Kontur des Greifers besteht der zweite Schritt darin, den Kreis zu detektieren. Dazu werden in dieser Arbeit diese folgenen Hauptfunktionen verwedent: Canny(), findContours(), medainBlur() und fitEllipse().

### 2.3.1 Canny-Algorithmus

Bei Bildern sind Kanten durch plötzliche Änderungen der Pixelintensität gekennzeichnet. Eine grafische Darstellung ist wie z. B. im Bild 3 gezeigt. Es ist offensichtlich, dass eine Änderung der Pixelintensität zwischen dem Gesicht und den Haare gibt.



Abbildung 3: Die Pixelintensitätsänderung der Kante[11]

Die Abbildung 4 zeigt den Pixelintensitätstrend eines 1D-Bildes, wobei die x-Achse die Zeit und y-Achse die Pixelintensität beschreibt. Eine Kante wird durch den „Sprung“ der Intensität wie in dem Bild angezeigt. Wird die erste Ableitung genommen, ist der „sprung“ besser zu erkennen. Eine Darstellung ist wie das Bild 5 gezeigt. Die Ableitung beschreibt die Änderungsgeschwindigkeit an einem Punkt, mit dem kann die Änderung der Pixelintensität in einem Bild gut repräsentiert.

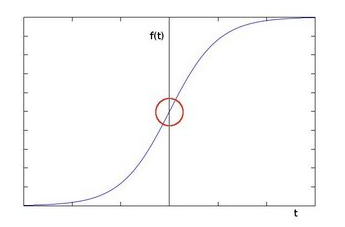


Abbildung 4: Der Pixelintensitätstrend eines 1D-Bildes[11]

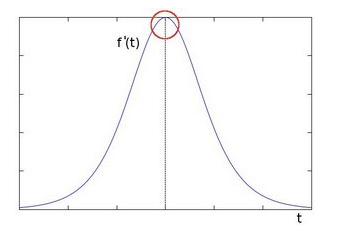


Abbildung 5: Die erste Ableitung der Intensitätsänderung[11]

In OpenCV gibt es mehrere Möglichkeiten um die Kanten zu detektieren z. B. Sobel-Operator, Laplace-Operator und Canny-Operator. Davon nutzt Sobel-Operator diese Idee. Der Sobel-Operator ist ein diskreter Differenzierungsoperator. Es berechnet eine Näherung des Gradienten einer Bildintensitätsfunktion in beiden x- und y-Richtung, was durch die Faltung mit einem Kernmatrix berechnet. Die größeren Kerne ergeben eine bessere Annäherung an die Ableitung. Die kleineren Kerne reagieren empfindlicher auf Rauschen.

In einem 2D-Bild wird die Faltung in einer Form von Matrix implementiert, siehe Abbildung 6. Die Werte an den entsprechenden Positionen der Matrix werden multipliziert und schließlich addiert. Die bekommende Summe ist der Wert des entsprechenden Pixels.

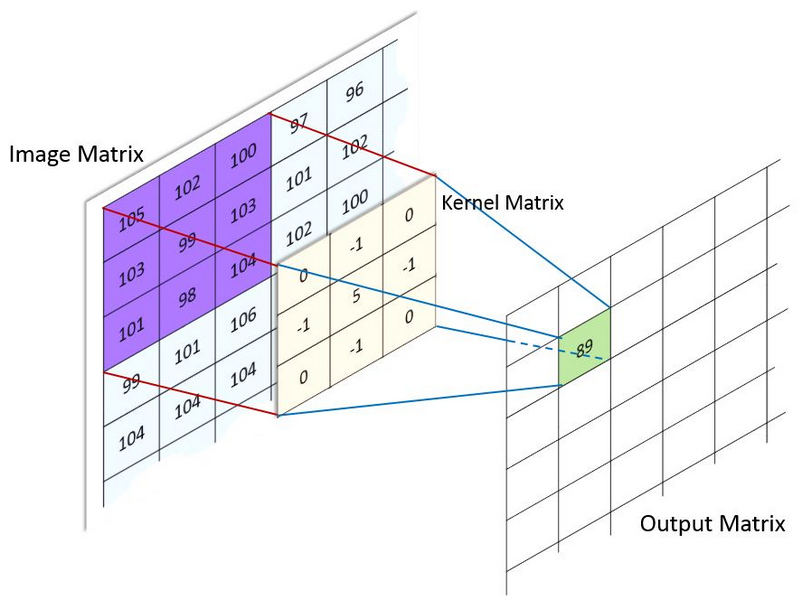


Abbildung 6: Die Faltung in einem 2D-Bild[12]

Ein Sobel-Operator ist einfach zu implementieren und schnell zu berechnen. Aber er reagiert empfindlich auf diagonale Kanten und erzeugt dicke Kanten, was die Genauigkeit einer weiteren Detektion beeinflusst.

Die Grundidee in Laplace-Operator um die Kanten zu detektieren ist ähnlich wie den Sobel-Operator. Der Laplace-Operator benutzt die zweite Ableitung der Intensitätsänderung eines Bildes. Wie vorher erwähnt, die Intensitätsänderung einer Kante wird dargestellt als einen „Sprung“ der ersten Ableitung. Das bedeutet, wird die erste Ableitung genommen, ist dieser „Sprung“ ein lokales Maximum. Wenn jetzt die zweite Ableitung genommen wird, ist es nicht schwer zu finden, dass der Wert des punktes, an dem ein Maximum sich befindet, in der zweiten Ableitung Null ist(siehe Abbildung 7). In Laplace-Operator wird diese Eigenschft der Ableitung verwendet, um die Kanten zu detektieren. Der Vorteil des Laplace-Operators besteht darin, dass er isotrop ist, d. h. er erkennt Kanten in allen Richtungen gleichermaßen und erzeugt dünne, scharfe Kanten. Der Nachteil ist, dass es sehr rauschempfindlich ist, da es kleine Änderungen der Pixelintensität verstärkt und außerdem keine Informationen über die Kantenausrichtung oder Ausrichtung liefert [13].

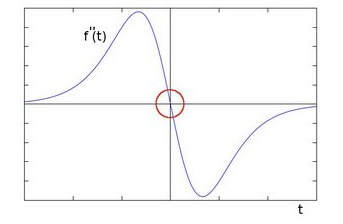


Abbildung 7: Die zweite Ableitung der Intensitätsänderung[14]

Canny Edge Detection ist ein beliebter Kantenerkennungsalgorithmus. Es wurde von John F. Canny entwickelt.

Die Canny() Funktion findet die Kanten in den Eingabebildern mit dem Canny Algorithmus. Canny-Edge-Detection-Algorithmus ist ein mehrstufiger Algorithmus, der die folgenden Stufen enthält: Rauschminderung, Ermitteln des Intensitätsgradienten des Bildes, Nicht-Maximum unterdrücken und Hysterese-Schwellenwert [15].

**Rauschminderung:**

Da die Kantenerkennung anfällig für Bildrauschen ist, besteht die erste Stufe darin, das Bildrauschen zu entfernen. Dies wird im nächsten Kapitel genauer besprochen.

**Ermitteln des Intensitätsgradienten des Bildes:**

Der Gradient eines Bildes ist einer der Grundbestandteile der Bildverarbeitung und Computer Vision. Er repräsentiert eine Richtungsänderung der Intensität oder Farbe in einem Bild. In diesem Schritt wird das vorher geglättete Bild mit einem Sobel-Kern gefiltert in beide horizontaler und vertikaler Richtung. Die erste Ableitung in der horizontalen und vertikalen Richtung ist damit ausgerechnet [16].

Die Richtung des Gradienten ist immer senkrecht zu den Kanten und sie wird auf einen von vier Winkeln gerundet: die vertikale, horizontale und zwei diagonale Richtungen. Eine grafische Darstellung ist wie folgend gezeigt. Die Winkel in [0°, 22,5°] oder [157,5°, 180°] wird auf 0° abgebildet.

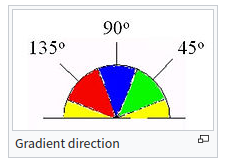


Abbildung 8: Die Richtungen des Gradienten[17]

**Nicht-Maximum unterdrücken:**

Wie der Name dieses Schritts werden im diesen Schritt die schärfsten Gradienten beibehalten und die anderen werden verworfen, womit eine Kantenverdünnung erreicht wird. Nachdem die Größe und Richtung des Gradienten ermittelt wurde, wird ein vollständiger Scan des Bildes durchgeführt, um alle unerwünschten Pixel zu entfernen, die möglicherweise nicht die Kante bilden. Dazu wird bei jedem Pixel geprüft, ob es in seiner Nachbarschaft in Gradientenrichtung ein lokales Maximum gibt, siehe Abbildung 9.

Abbildung 9: Unterdrückung nicht maximales Werts[18]

Der Punkt A liegt auf einer Kante. Die Richtung des Gradienten ist vertikal zu der Kante. B und C sind zwei Punkte, die auch auf der Richtung des Gradienten liegen. In diesem Fall wird der Punkt A mit Punkt B und C überprüft, um zu sehen, ob er ein lokales Maximum bildet. Wenn ja, bleibt der Punkt A erhalten in die nächste Stufe, andernfalls wird es unterdrückt (auf Null gesetzt). Das Ausgabebild ist damit binäres Bild [19].

**Hysterese-Schwellenwert:**

Auf dieser Stufe wird entschieden, welche Kanten wirklich Kanten sind und welche nicht. Das „Hysteresis’ hier bedeutet die Abhängigkeit des Zustands eines Systems von seiner Geschichte. In der Canny() Funktion kann es als die Abhängigkeit eines Pixels von seinen Nachbarn verstanden werden. Das wird im folgenden erklärt werden. Um zu unterscheiden, ob es sich um eine reale Kante handelt werden in Canny() zwei Schwellenwerte verwendet werden: minVal und maxVal, welche bedeuten der minimale Wert und der maximale Wert. Die Kante mit einem Intensitätsgradienten größer als maxVal werden als reale Kanten betrachtet und diejenigen unterhalb von minVal werden keine Kanten betrachtet und werden daher verworfen. Diejenigen, die zwischen diesen beiden Schwellenwerten liegen, werden nach ihrer Konnektivität als Kanten oder Nichtkanten klassifiziert. d.h. wenn sie mit „sicheren Kanten“-Pixeln verbunden sind, werden sie als Teil von Kanten betrachtet. Andernfalls werden sie ebenfalls verworfen [20].

Siehe das Bild unten:

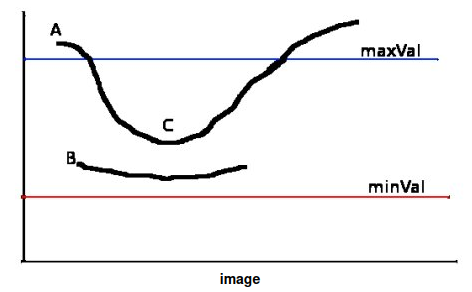


Abbildung 10: Hysterese-Schwellenwert[18]

Die Kante A liegt über dem maxVal und wird daher als reale Kanten betrachtet. Obwohl Kante C unter maxVal liegt, ist sie mit Kante A verbunden, deswegen wird sie auch als gültige Kante betrachtet. Ebenso ist der obere rechte Teil von Kante C, der über dem maxVal liegt. Daher erhalten wir die vollständige Kurve. Jetzt betrachten wir die Kante B. Obwohl sie über minVal liegt und sich im gleichen Bereich wie Kante C befindet, ist sie mit keinen realen Kanten verbunden und wird folglich verworfen. Daher ist es von wesentlicher Bedeutung, dass wir das geeignete minVal und maxVal auswählen, um das richtige Ergebnis zu erhalten. Außerdem werden in dieser Phase die kleine Pixelrauschen auch entfernt, da es davon ausgegangen wird, dass es sich bei den Kanten um lange Linien handelt. Und was wir endlich erhalten sind starke Kanten im Bild.

Durch den Prozess des Canny-Edge-Detection-Algorithmuses ist es zu sehen, dass Canny-Algorithmus dünne und präzise Kanten erzeugen kann. Außerdem kann er sich durch einstellbare Parameter an unterschiedliche Bildeigenschaften anpasst werden. Die Nachteile bestehen darin, dass er relative komplexer und rechenintensiver ist und eine manuelle Abstimmung der Parameter erfordert, um optimale Ergebnisse zu erzielen [18].

Trotz der komplexen Schritte des Canny-Edge-Detection-Algorithmuses, bietet er eine vergleichsweise höher Genauigkeit, wird es in dieser Arbeit dieser Algorithmus verwendet.

### 2.3.2 Konturenerfassung

In der Bilderarbeitung werden die Konturen häufig verwendet zur Formanalyse sowie zur Objektdetektion und -erkennung. Konturen können einfach als eine Kurve erklärt werden, die alle angrenzenden Punkte (entlang der Grenze) derselben Farbe oder Intensität verbindet. Die Funktion findContours() wird verwendet, um die Konturen in einem Bild zu finden. In früheren Versionen von OpenCV modifiziert diese Funktion das Oringinalbild. Seit OpenCV 3.2 wird diese Charakteristik gelöscht. In OpenCV, das Finden von Konturen bedeutet das Finden weißer Objekte vor schwarzem Hintergrund. Deswegen soll das zu findende Objekt weiß und der Hintergrund schwarz sein. Da das Ausgabebild von Canny() ein binäres Bild ist, braucht es hier nicht weiter zu bearbeiten.

Die Funktion findContours() gibt zwei Parameter zurück. Einer davon ist „contours“. Der Python-Typ von diesem Output ist „list“. Es enthält alle Konturen, die von dieser Funktion detektiert werden. Jede individuelle Kontur ist repräsentiert durch einem Numpy-Array von (x,y)-Koordinaten aller einzelnen Punkte des detektierten Objektes.

Ein anderes Output von findContours() ist „hierarchy“. Wenn die Objekte in einem Bild detektieret werden, liegen in manchen Fällen einige Formen innerhalb anderer Formen. In diesem Fall bezeichnen wir das Äußere als Eltern und das Innere als Kind. Auf diese Weise stehen die Konturen in einem Bild in einer gewissen Beziehung zueinander. Und wir können angeben, wie eine Kontur miteinander verbunden ist, z. B. ob sie einer anderen Kontur untergeordnet ist oder ob sie ein übergeordnetes Element ist usw. Die Darstellung dieser Beziehung wird als Hierarchie bezeichnet. In OpenCV ist diese Beziehung zwischen Konturen als ein Array aus vier Werten repräsentiert: [Next, Previous, First\_Child, Parent]. „Next“ bezeichnet den Index der nächsten Kontur auf derselben Hierarchieebene. „Previous“ bezeichnet den Index der vorherigen Kontur auf derselben Hierarchieebene. „First\_Child“ bezeichnet den Index seiner ersten untergeordnete Kontur und „Parent“ bezeichnet den Index seiner übergeordneten Kontur. Wenn eine solche Kontur existiert nicht, wird dieses Feld als -1 angenommen. Wird die Aufgabe unserer Arbeit betrachtet, ist dieser Parameter nicht relevant, da nur eine der Konturen der Kreise benötigt wird. Dieser Parameter wird deswegen nicht weiter besprochen und berücksichtigt [22].

FindContours() hat insgesamt sechs Parameter. Einer davon ist das Eingabeparameter „image“, das ein 8-Bit-Einkanalbild fordert. Pixeln ungleich Null in dem Eingabebild werden als Einsen behandelt. Nullpixeln bleiben Nullen, daher wird das Bild als binär behandelt. Die andere Parameter dieser Funktion sind „contours“, „hierarchy“, „mode“, „method“ und „offset“, wobei „contours“ und „hierarchy“ Ausgaben sind. Der Parameter „mode“ legt den Modus des Konturabrufs. In opencv 4.7.0 gibt es vier Kontur-Abruf-Modi: RETR\_EXTERNAL, RETR\_CCOMP, RETR\_LIST und RETR\_TREE.

Der RETR\_EXTERNAL Modus gibt nur die extreme äußeren Konturen zurück und besitzt die schnelleste Geschwindugkeit. Im unseren Fall wird es nich in Betracht kommen. Der RETR\_CCOMP Modus ruft alle Konturen ab und ordnet sie in einer zweistufigen Hierarchie an. Die äußere Konturen des Objekts (d. h. seine Grenze) werden in Hierarchie 1 platziert. Und die Konturen der Löcher im Objekt, bzw. die innere Konturen (sofern vorhanden) werden in Hierarchie 2 platziert. Wenn sich darin ein Objekt befindet, wird dessen Kontur wieder in Hierarchie 1 platziert. Und wenn es darin ein Loch gibt, wird es wiede in Hierarchie 2 platziert und so weiter. Der RETR\_LIST Modus ruft alle Konturen ab, ohne hierarchische Beziehungen herzustellen， d. h. alle Konturen gehoeren zur gleichen Hierarchieebene. Hier ist also der 3. und 4. Term im Hierarchie-Array immer -1. Wie den RETR\_LIST Modus ruft der RETR\_TREE Modus alle Konturen ab. Außerdem erstellt es eine vollständige hierarchische Beziehungen und besitzt es deswegen die langsamste Geschwindigkeit. Wie vorher gesprochen, die Hierarchie der Konturen eines Bildes ist nicht relevant in unserer Aufgabe und deshalb passen alle letzten drei modi zu dieser Aufgabe. Für eine geringere Menge der Datenverarbeitung wird in dieser Aufgabe der Modus RETR\_LIST benutzt.

Der Parameter „method“ legt die Methode zur Konturnäherung fest. Die vorhandenen Konturnäherungsmethoden sind: CHAIN\_APPROX\_NONE, CHAIN\_APPROX\_SIMPLE, CHAIN\_APPROX\_TC89\_L1, CHAIN\_APPROX\_TC89\_KCOS.

Die Methode CHAIN\_APPROX\_NONE speichert alle Konturpunkte da hier keine Konturnäherungsmethoden verwendet werden.

Die Methode CHAIN\_APPROX\_SIMPLE entfernt alle überflüssigen Punkte und komprimiert die Kontur. Wie in der Abbildung 11 dargestellt wird die Kontur eines Rechtecks mit CHAIN\_APPROX\_NONE und CHAIN\_APPROX\_SIMPLE detektiert. Es ist offensichtlich zu sehen，dass nur die vier Endpunkte benötigt werden um ein Rechteck zu beschreiben. Dadurch wird der Speicherplatz erheblich gespart.

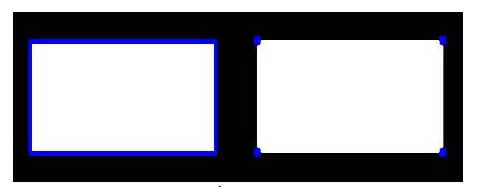


Abbildung 11: Detektierte Konturpunkte mit CHAIN\_APPROX\_NONE(links) und CHAIN\_APPROX\_SIMPLE(rechts)[22]

Die Methode CHAIN\_APPROX\_TC89\_L1 und CHAIN\_APPROX\_TC89\_KCOS sind verwendet, wenn die Formen gekrümmt und keine einfachen Polygone sind, wobei Berechnungen mit höherer Präzision erfordern [23].

Wenn diese zwei Methoden mit vorherigen Parameter verwendet werden, werden keine Ellipsen detektiert. Daraus ist es zu entnehmen, dass in unserem Fall da die zu detektierte Form eine einfache Ellipse ist, sind diese zwei Methode nicht geeignet.

Der letzte Parameter von findContours() ist „Offset“. Er ist ein optionaler Parameter, mitdem jeder Konturpunkt verschoben werden kann. Es ist verwendet um die Konturen, die aus dem Bild-ROI extrahiert werden, in dem Kontext des gesamten Bild zu analysieren. Dieser Parameter ist für diese Aufgabe nicht relevant, deswegen wird er in dieser Arbeit nicht weiter berücksichtigt.

### 2.3.3 Kreisdetektion

Wie in Abbildung 12 zeigt, ist das Modell des Greifers in der Simulationsumgebung. Der von dem roten Kreis ausgezeichnete Teil ist der Greiferkopf, der der zu detektierenden Teil ist.

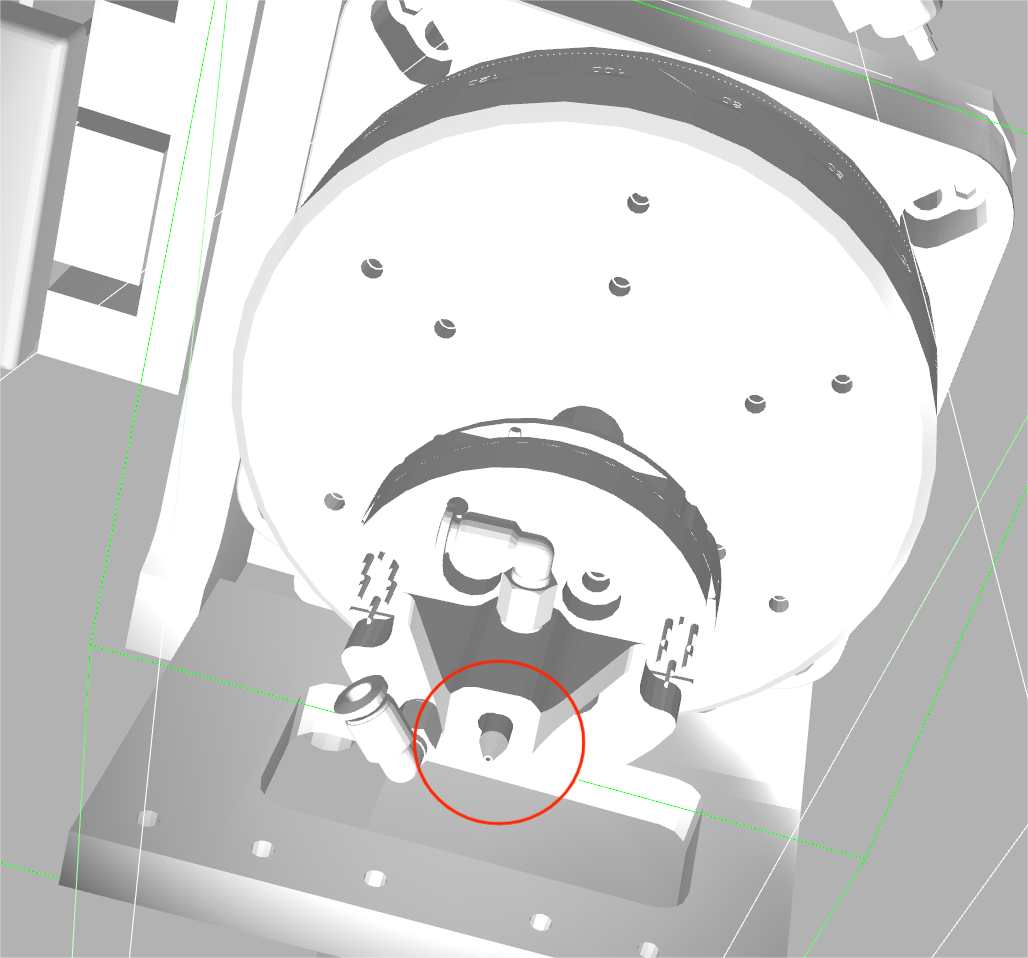


Abbildung 12: Das Greifersmodell in der Simulation

Wird er mit der Unterseitenkamera fotografiert, wird die Abbildung 13 bekommen. Darunter ist der graue Bereich die Neigungsebene des Greiferkopfs und der weiße Ringbereich ist seine untere Ebene. Relevant für die Detektion ist der innere Kreis des Ringbereich. Dies erfolgt mit der Programmbibliothek OpenCV. Dazu wird die Algorithmus zur Ellipsendetektion in OpenCV verwendet, wobei der Kreis als eine Ellipse detektiert wird. Wenn die Hauptachse und die Nebenachse einer Ellipse nahezu gleich lang sind, kann diese Ellipse als ein Kreis betrachtet. Die Gültigkeit dieser Methode wird im Abschnitt 3.4.3 geprüft.

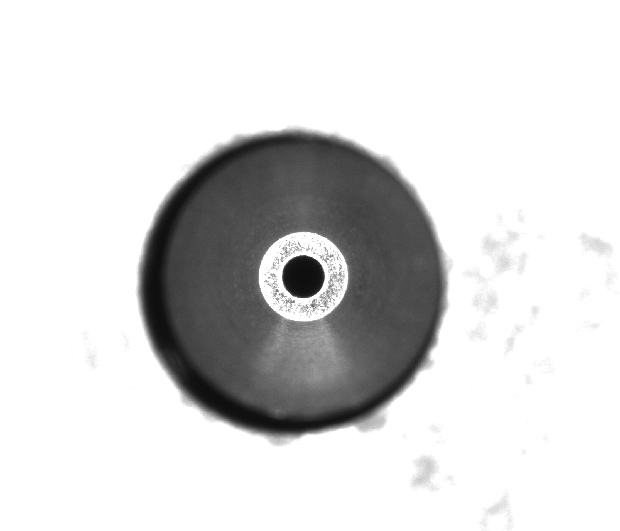


Abbildung 13: Das mit der Kamera aufgenommene Greifersbild

In OpenCV gibt es insgesamt drei Funktionen um eine Ellipse zu detektieren: fitEllipse(), fitEllipseAMS() und fitEllipseDirect(). Alle diese drei Methoden berechnen die Ellipse, die zu einer Reihe von 2D-Punkten passt. Der Unterschied liegt an die verwendeten Methoden. fitEllipse() rechnet die Algebraische Distanz, fitEllipseAMS() benutzt die Methode „Approximate Mean Square“, und fitEllipseDirect() benutzt die Methode „Direct Least Square“.

Die Eingabe und Ausgabe aller drei Funktionen sind gleich. Die Eingabe „points“ ist eine Menge von 2D-Punkten. Hier kann das Ausgabeparameter „contours“ von findContours() direkt verwendet werden. Die Ausgabe „retval“ ist die detektierte Ellipse, deren Python-Typ „tuple“ ist. In dieser „tuple“ gibt es drei Elemente. Das erste Element beschreibt den Mittelpunkt der Ellipse. Das zweite Element ist Länge der Hauptachse und der Nebenachse, wobei der Wert ist doppelt. Der Letzte ist der Drehwinel, der in unserer Aufgabe nicht relevant ist.

# 3. Arbeitsdurchführung

## 3.1 Kalibrierungsvorgehensweise

Die Kalibrierungsvorgehensweise besteht hauptsätzlich aus drei Teilen: das Steuern des Greifers, die Detektion des Kreises sowie die Auflösung der Mitte der roten Platte. Wie im obigen gesagt, auf Grund der Montagertechnologie des Greifers wird eine radiale Verschiebung zwischen dem Greifer und der Platte entstehen. D. h. der Greifer und die rote Platte nicht koaxial sind. Da die Antriebsachse der Rotation die rote Platte ist, dreht sich der Greifer mit der Platte. Im idealfall dreht sich der Greifer um seine Mitte bzw. die Achse des Mittelpunktes der roten Platte und sie drehen zusammen um diese gleiche Achse. Sind sie nicht koaxial, wird der Greifer eine kreisförmige Bewegung um die Achse machen. Wie in Abbildung 13 dargestellt, ist die Bewegung des Greifers in der tatsächlichen Situation. Der kleine rote Bogen bzw. die Punktlinie ist die Bewegungsspur der Mitte des Greifers, wobei die Mitte der roten Platte auf der zu der Bildebene fast senkrechten Mittellinie der kreisförmigen Punktlinie liegen soll. Die zu suchende Abweichung in dieser Arbeit ist der Abstand zwischen den zwei Mittelpunkten auf der Ebene bzw. der Radius der kreisförmigen Punktlinie.

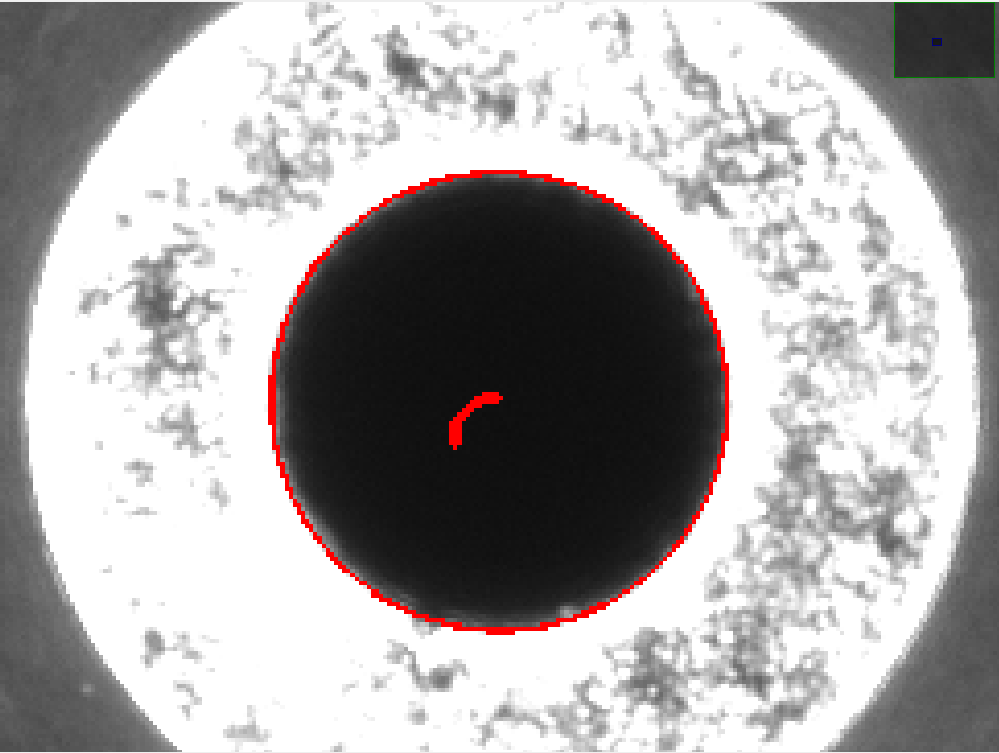


Abbildung 14: Die Bewegungsspur der Mitte des Greifers

Um diesen Radius zu berechnen, ist der erste Schritt das Steuern des Greifers zur bestimmten Position, die liegt ungefähr im Mitte der Kamerasicht. Dann wird der Greifer rotiert und dabei werden alle Mittelpunkte des Greifers gesammelt. Mit diesen Punkten werden anschließend der Radius der roten Platte sowie die relative Position des Greifers zur Platte ausgerechnet. Schließlich wird eine automatische Anpassung der bestimmten Datei mit diesen ausgerechneten Daten erreicht.

## Hauptlogik des Programms

In dieser Aufgabe wird es angefordert, dass eine Kalibrierung in beiden Realität und Simulation durchgeführt werden kann. Deswegen werden die unterschiedliche Parameter benötigt um es in unterschiedlichen Situationen laufen zu können.

Nachdem das Programm beginnt, folgt eine While-Schleife. Dadurch kann zwei Kalibrierungsmodi, nämlich die Kalibrierung in der Realität und die in der Simulation ausgewählt werden, und dabei wird es immer in diese Auswahlschleife gehen wenn der eingegebene Modus keiner dieser beiden ist. Wird ein richtiger Modus ausgewählt, wird dann die entsprechenden Parameter verwendet.

Um das Roboter zu steuern sowie seinen Zustand zu überwachen ist die Instanziierung der Class ActionClient() und die Verwendung der Funtion create\_subscription() benötigt. Mithilfe der Class CvBridge() kann das in ROS aufgenommene Bild in OpenCV verarbeitet werden. Außerdem wird eine Funktion create\_timer() verwendet, um einen Timer zu erstellen, sodass eine regelmäßige Prüfung des bestimmten Werts erreicht wird.

In der Callback-Funktion der Funtion create\_subscription() wird die Ausführungsreihenfolge der Funktionen angegeben. Der erste Schritt ist zu prüfen, ob der Greifer an der angegebenen Position im Sichtfeld der Kamera ankommt. Entsprechend der Genauigkeit des Controllers wird eine Toleranz zwischen -0.000001 und +0.000001 eingestellt. Wenn der Greifer innerhalb dieses Bereichs liegt, gilt es als angekommen. Außerdem muss die erwünschten Positionen desired.positions die angegebenen Positionen entsprechen, um eine logische Verwirrung der Ausführungsreihenfolge zu vermeiden. Wird ein Gelenk die entsprechende Position erreicht, wird die Variable self.reached\_joint\_number um eins erhöht. Wird alle Gelenke die Bedingungen erfüllen, wird ein ROS-Parameter „ok“ erstellt und seinen Wert auf eins eingestellt. Dieser Wert wird durch den Timer regelmäßig prüft. Wenn der Wert eins ist, werden die Detektion des Greifers und dessen Rotation startet. Endet die Rotation, wird die Detektion ebenso beendet und eine darauffolgende Ausrechnung wird startet. Das Rechnungsergebnis wird grafisch dargestellt und mit der Taste „s“ beendet und eine Anpassung der Datei wird danach erfolgt.

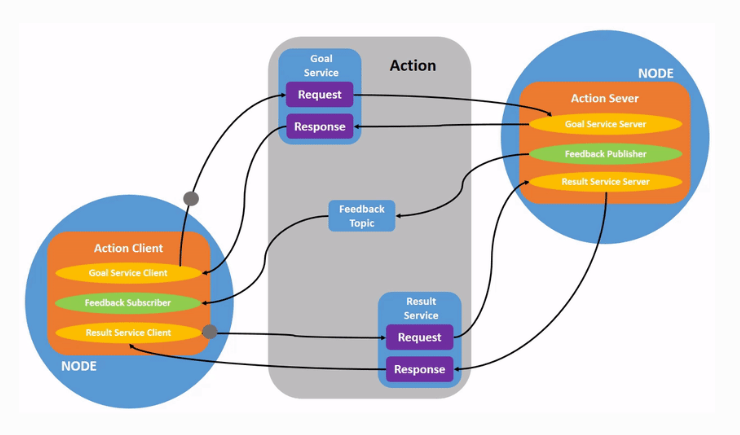
## 3.3 Steuern der Trajektorie

### 3.3.1 Steuerungsmethode

Um die Trajektorien an den Controller zu senden kann man zwei Mechanismen davon benutzten: „Action Interface“ oder „Topic Interface“. „Topics“ werden normalerweise für kontinuierliche Datenströme wie Sensordaten, Roboterstatus usw. verwendet, wobei der Datenfluss unidirektional ist. „Actions“ werden meistens für diskretes Verhalten verwendet, das zur Steuerung eines Roboters oder zum Laufen über einen längeren Zeitraum. Während der Ausführung der „Actions“ werden die Feedbacks gegeben. Im unseren Fall wird ein Präzisionsmontageroboter gesteuert und die Feedbacks während dem Prozess sind erwünscht, deswegen wird in dieser Arbeit die Steurung des Roboter durch die Kommunikation mit „Action“ erreicht.

Für die Aufgabe dieser Arbeit ist das Steuern durch die „Action“-Kommunikation besser geeignet wegen der Möglichkeit des Debuggings während der Bewegung. Deshalb wird das Pragramm zur Steuerung des Greifers in dieser Arbeit rund um diesen Kommunikationsmodus erweitert.

Unten ist eine grafische Darstellung der „Action“-Kommunikation. Der „Action Client“ fordert ein Request an den „Action Server“ an und der „Action Server“ sendet eine Antwort zurück, um es aufzuweisen, ob das Request akzeptiert wird. Wenn das Request erfolgreich akzeptiert wird, wird das Request bzw. die „Action“ ausgeführt. Im Anschluss an der „Request“-Anforderung stellt der Client eine asynchrone Anforderung für das Ergebnis. Während das Ziel ausführt wird, gibt der „Action Server“ die kontinuierlichen Feedbacks an den „Action Client“. Und letztendlich gibt der „Action Server“ ein einmaliges Feedback, um zu deuten ob die „Action“ erfolgreich beendet wird.

Abbildung 15: Die „Action“-Kommunikation[7]

Genauso wie die Funktionsweise der „Action“-Kommunikation wird das Programm des Steuerns des Greifers implemeiert.

Zuerst wird die Trajektorie definiert. Um einen unmittelbaren Hinweis zu geben, mit dem ein Startsignal der Funktion mitgeteilt wird, wird eine Info „Starting align…“ gegeben. Anschließend folgt die Definition der Bewegung. Die Trajektorie ist angegeben durch die Message „trajectory\_msgs/msg/JointTrajectoryPoint“ [24], die wie folgend definiert:

double[] positions

double[] velocities

double[] accelerations

double[] effort

builtin\_interfaces/msg/Duration time\_from\_start

Die erste vier Elemente verwenden den Datentyp „double“. „double“ ist ein DDS-Typ, der dem ROS-Typ „float 64“ und dem Python-Typ „float“ entspricht. Der Datentyp des letzten Elementes kann durch den ROS-Typ „int32“ oder „uint32“ oder Python-Typ „int“ angegeben. „position“ gibt an, wo sich die Zielposition befindet [25]. Die Zielposition ist so gewählt, dass das „SPT\_Vacuumtool“ ungefähr in der Mitte des Kamerasichtfeldes steht, womit eine spätere Rotation erfolgreich ausgeführt werden kann. Außerdem soll die Höhe des „SPT\_Vacuumtool“ richtig gewählt werden, sodass es in der Bildverarbeitung besser behandelt werden kann. Bei der realen Anlage wird die Zielposition auf [-0.359, -0.0458, 0.03, 0.0] eingestellt und in der Simulationsumgebung wird sie auf [-0.359, -0.0458, -0.051544, 0.0] eingestellt. Beide Werte werden getestet und sind in der entsprechenden Situation sinnvoll. „velocities“ und „accelerations“ stellen es fest, welche Werte die Geschwindigkeit und die Beschleunigung sind wenn das Robot an dem Zielpunkt ankommt. „effort“ beschreibt das Drehmoment oder die Kraft, die an jedem Gelenk ausgeübt werden. In unserem Fall ist dieser Wert nicht relevant. Das letzte Element „Duration“ gibt die gewünschte Zeit vom Start der Trajektorie bis zum Erreichen des Zielnpunkts. Für eine stabile Bewegung, wird dieser Wert auf 6 Sekunden eingestellt. Dieser Wert ist nur in der Simulationsumgebung relevant. Um eine Trajektorie richtig einzustellen, muss es nur einer von „velocities“ und „Duration“ ausgewählt wird. Andernfalls wird die Bewegung des Greifers fehlerhaft wegen eines Widerspruchs dieser zwei Werte zueinander.

Nachdem die Trajektorie definiert wird, muss sie dann an den Roboter gesendet werden.

Wie oben erwähnt, wird in dieser Arbeit die „action“-Kommunikation benutzt zum Senden der Trajektorie. Nach Eingabe des Befehls „ros2 action list -t“ in dem Terminal werden die vorhandenen Aktionen inklusive der Namen ihrer Schnittstellen angezeigt. Es ist dann offensichtlich eine „Action“ heißt „/joint\_trajectory\_controller/follow\_joint\_trajectory“ existiert. Ihre Schnittstelle ist „control\_msgs/action/FollowJointTrajectory“ [26]. In dieser Schnittstelle sind folgende vier Elemente relevant, um den Roboter zu steuern:

trajectory\_msgs/JointTrajectory trajectory

JointTolerance[] path\_tolerance

JointTolerance[] goal\_tolerance

duration goal\_time\_tolerance

Davon wird nur das Element „trajectory\_msgs/JointTrajectory trajectory“ benutzt. Die andere sind die unterschiedlichen Toleranz, um die Trajektorie weiter zu begrenzen. Werden sie nicht manuell angegeben, werden die Werte auf die Standardwerte für den „Action“-Server gesetzt. Dies Element ist wie folgend definiert:

Header

string[] joint\_names

JointTrajectoryPoint[] points

Es besteht aus drei Elemente: „Header“, „joint\_names“ und „points“, wobei points vorher definiert wird. Das Element „Header“ hat zwei Felder: „stamp“ und „frame\_id“. Diese zwei Elemente sind nicht relevant für die Definition einer Trajektorie und werden deswegen nicht weiter diskutiert. „joint\_names“ gibt an, welche Gelenke zu steuern sind. Im ersten Schritt müssen vier Gelenke gesteuert werden: „X\_Axis\_Joint“, „Y\_Axis\_Joint“, „Z\_Axis\_Joint“, „T\_Axis\_Joint“. Diese Namen müssen den, die in der Datei „pm\_robot.xacro“ dediniert werden entsprechen. „pm\_robot.xacro“ ist eine Datei, die die Konstruktion des Roboters beschreibt. Um das einzelne Gelenk zu steuern, was in dem folgenden Teil dieser Arbeit nötig ist, muss dazu das Parameter „allow\_partial\_joints\_goal“ hinzugefügt und dessen Wert auf „True“ eingestellt werden. Sonst müssen alle vier Gelenke definiertet werden, was zu einer Bewegung dieser alle vier Gelenke und einer misslungenen Detektion führt.

Schließlich werden die Feedbacks der Bewegung definiert, womit eine vollständige Bewegung der Positionierung beendet.

### 3.3.2 Programmierteil

Es wird die Methode get\_logger() des Nodes und dessen Methode info() verwendet um den Startsignal von align\_action() zu senden. Danach wird die Class JointTrajectoryPoint() instanziiert. Die vorher ausgewählten Parameter werden hier übergeben. Wird ein Steuern in der Simulation durchgeführt, wird das Attribut time\_from\_start benutzt. Dann wird die Class FollowJointTrajectory.Goal() instanziiert um die Trajektorie zu seden. Darauf folgt die Definition der Attribute joint\_names und points der Class. Um das Ziel erfolgreich zu senden, wird die Methode wait\_for\_server() der Class ActionClient() verwendet. Diese Methode wartet darauf, dass der Aktionsserver verfügbar ist, und sendet dann ein Ziel an den Server. Dann wird die Methode send\_goal\_async() der Class ActionClient() verwendet. Diese Methode sendet ein Ziel und erhält asynchron das Ergebnis, das als ein „Future“ zurückgegeben. Das Ergebnis wird auf ein „ClientGoalHandle“ gesetzt, wenn der Empfang des Ziels von einem Aktionsserver bestätigt wird. Der Hauptteil des Programms, das die Trajektorie sendet, ist hier fertig. Um zuletzt die Rückmeldung zu bekommen, ist es benötigt, die „callback“-Funktion zu addieren. Das erfolgt durch die Methode add\_done\_callback() der „Future“. Mit dieser Methode kann eine „callback“-Funktion eingestellt, die sofort aufgerufen werden kann, wenn die „Future“ erledigt ist. Ausserdem wird eine „callback“-Funktion benötigt, um die Rückmeldungen während den Prozess zu bekommen.

### 3.3.3 Rotationstrajektorie

Nachdem die Positionierung des Greifers fertiggemacht wird, muss eine Rotation folgen, sodass die Detektion erfolgreich durchgeführt werden kann.

Um möglichst viele Punkte zu sammeln bzw. eine möglichst bessere Interpolation des Kreises zu machen und auch gleichzeitig die maximale Rotationsgrenze des Greifers zu berücksichtigen, wird die Rotation auf -1300000 Inkrement eingestellt, wobei 100 Inkrement 0.01° entspricht. Der Drehwinkel beträgt deswegen ca. 130°. Eine Drehung in einem größeren Winkel kann das mit dem Greifer verbundene Kabel beschädigen.

Die Logik und der Code für das Rotieren sind fast gleich wie die Positionierung des Greifers und werden deswen nicht mehr besprochen.

## Detektion des Greifers

### 3.4.1 Rauschminderung mit Median-filter

Vor der Detektion eines Bildes wird es häufig zuerst eine Vorbehandlung impementiert. Damit kann Eine bessere Qualität der späteren Detektion erreicht werden.

Bevor dieser Abschnitt beginnt, wird ein vergrößertes Greifersbild betrachtet. Es ist klar zu sehen, dass es eine Lücke an der unteren Ebene gibt, siehe Abbildung 16. Dies liegt an der Verarbeitungstechnologie der Werkzeugmaschine.

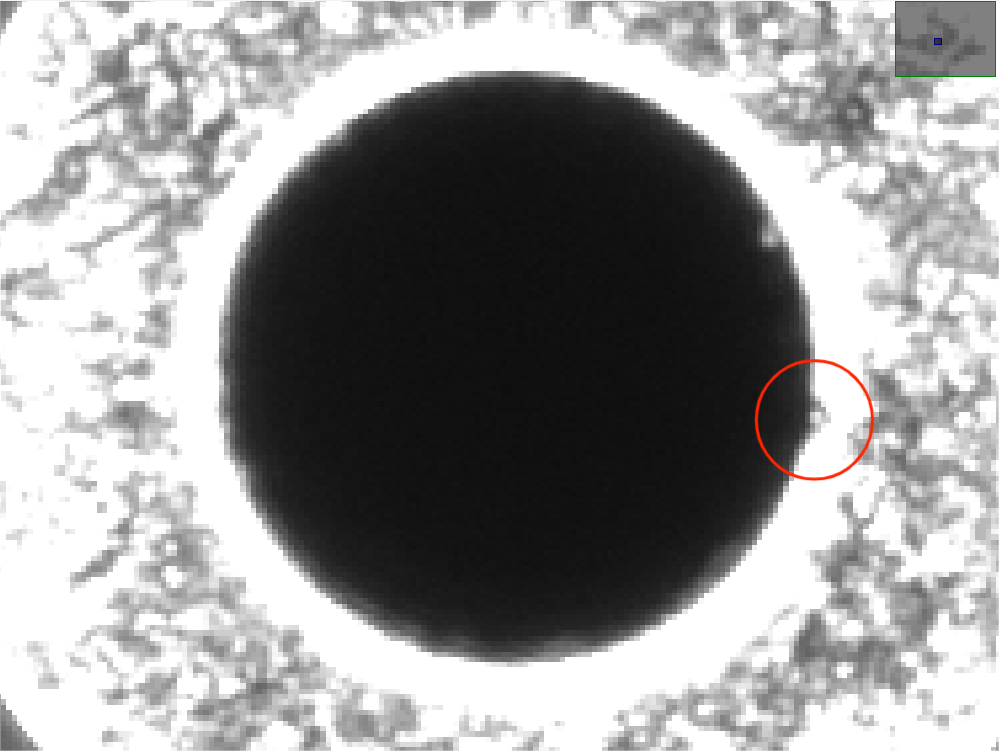


Abbildung 16: Die Lücke an der unteren Ebene

Wird diese Kante detektiert und dann damit eine interpolation impementiert, wird ein Fehler auftreten.

Der Grund liegt daran, dass die Ellipse, die von der Funktion fitEllipseDirect() interpoliert wird, eine global optimierte Lösung ist. D. h. jeder Punkt an der Kante hat eine Beeinflussung zu dem endlichen Ergebnis.

Wird der Greifer unter einer dunkler Beleuchtung fotographiert, ist dieser Fehler klarer zu sehen, siehe Abbildung 17. Abbildung 18 zeigt das entsprechende Detektionsergebnis. Es ist zu sehen, dass wegen der Beeinflussung der Punkte an der Lücke liegt die Kante der Ellipse hier außer der realen Kante.

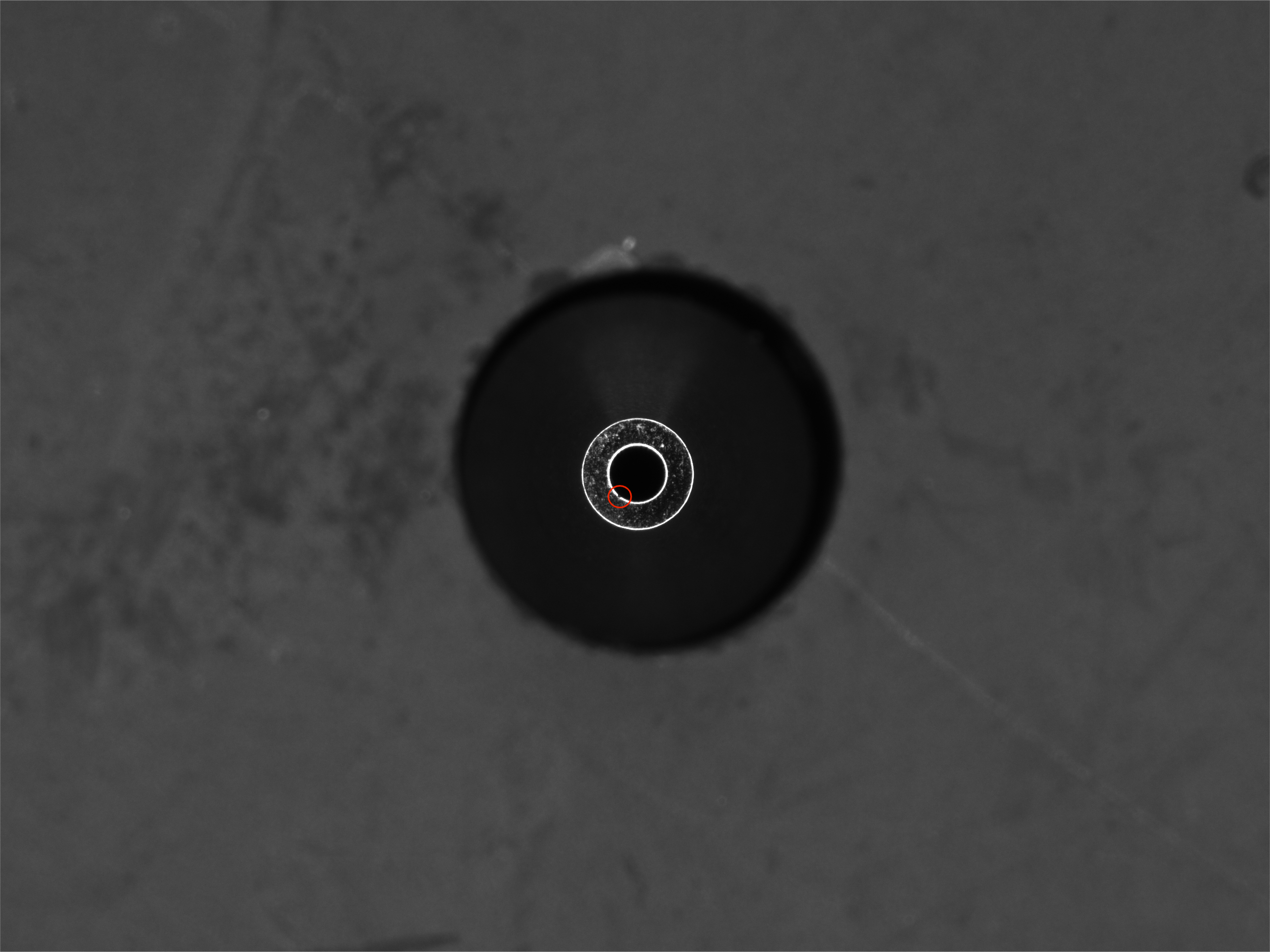


Abbildung 17: Die Lücke unter dunkler Beleuchtung



*Abbildung 18: Das Detektionsergebnis*

In der realen Detektion wird dieser Fehler viel verkleinert wegen einer stärkeren Beleuchtung. Dieser Punkt kann trotzdem nicht vernachlässigt werden.

Nun wird noch einmal die Abbildung 13 betrachtet, ist es offensichtlich zu sehen, dass es viele schwarze Punkte auf der unteren Ebene des Vakuumgreifers bzw. auf dem weißen Ringbereich. Dies liegt an der Rauheit der unteren Ebene. Das Merkmal von diesem Rauschen ist ähnlich wie das Salz-und Pfefferrauschen. Das Salz-und Pfefferrauschen, auch Impulsrauschen genannt, ist eine Form von Rauschen. Dieses Rauschen kann durch scharfe und plötzliche Störungen im Bildsignal verursacht werden. Es zeigt sich als spärlich vorkommenden weißen und schwarzen Pixeln. Typischweise ist ein Median-Filter eine wirksame Methode zur Unterdrückung für diese Art von Rauschen.

Die Medianfilterung wird in der Bildverarbeitung häufig verwendet, da sie Kanten gut bewahrt und gleichzeitig Rauschen entfernt kann. Die Abbildung 19 zeigt die Funtionsweise eines 3 × 3 Median-Filters.

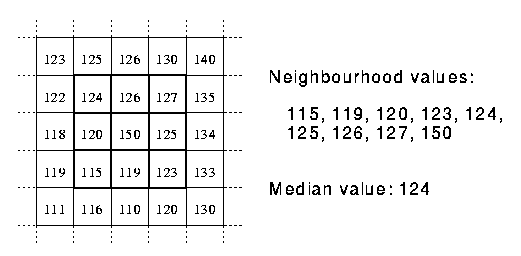


Abbildung 19: Die Funtionsweise eines 3 × 3 Median-Filters[27]

Mit einem Median-Filter werden zuerst alle mitrechnenden Pixeln aufgelistet. Dann wird das mittlere Wert als den Wert dieses Pixels genommen.

In OpenCV wird die Funktion medianFlur() verwendet dafür. Diese Funktion hat insgesamt drei Parameter: „src“, „kzise“ und „dst“. „src“ ist ein Eingabebild und „dst“ ist das Ausgabebild. „ksize“ gibt an, wie groß die Blendegröße ist.

Zum besseren Vergleich und Erklärung wird es zuerst ein Bild ohne Verarbeitung dargestellt, siehe Abbildung 20.

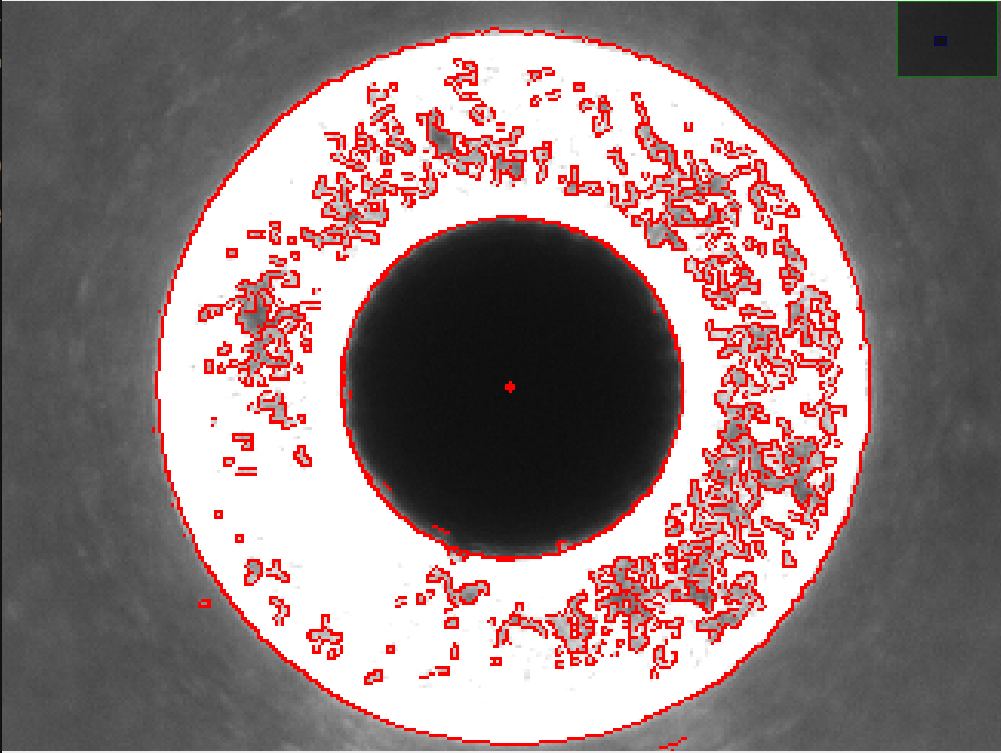
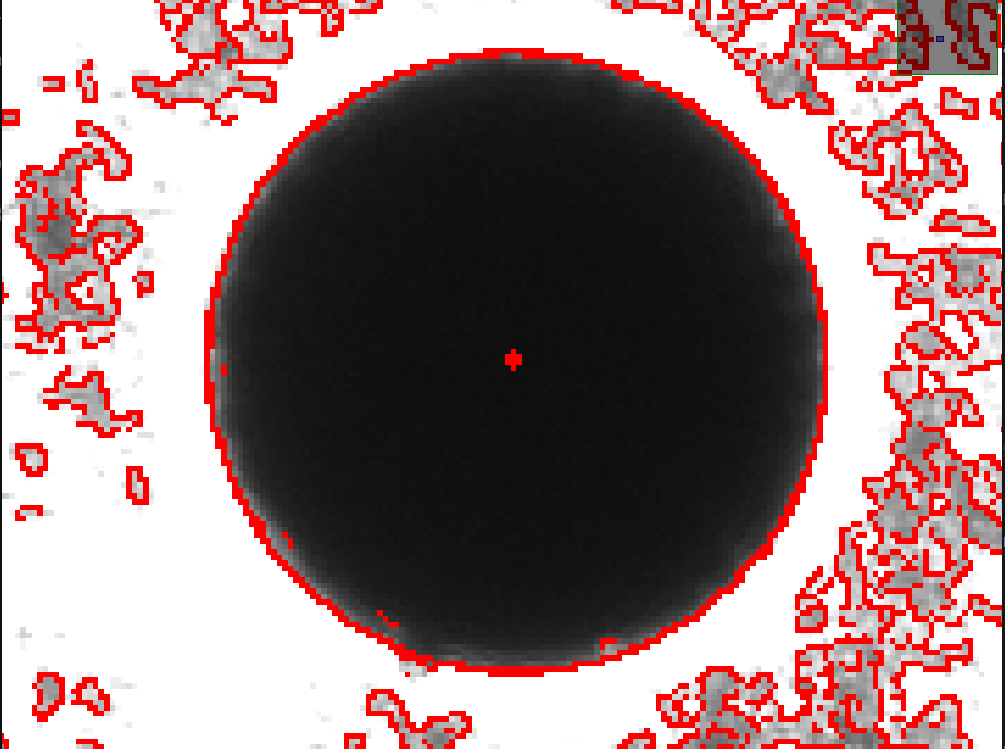
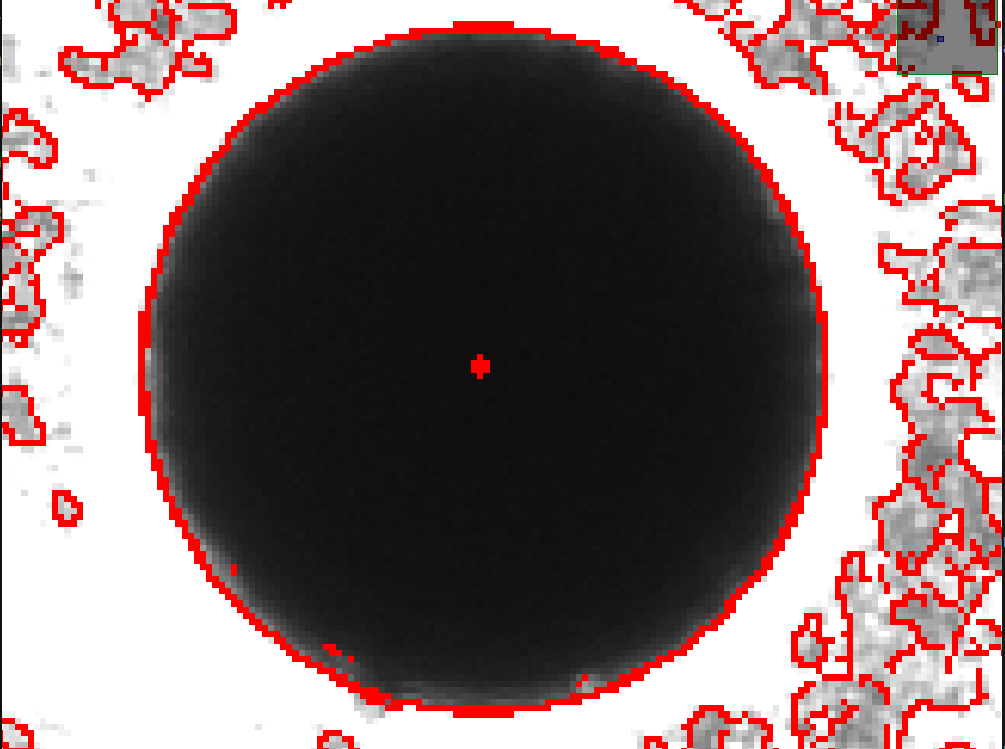


Abbildung 20: Detektion ohne Bildverarbeitung

Es ist aus der Abbildung ersichtlich, dass es viele Rauschen gibt. Die detektierten Kanten passen dennoch gut zu den Kreise.

Unten sind die Bilder, die die Auswirkungen von medianBlur() auf die Detektion mit der Blendegröße 3, 5, 7, 9 zeigen. Eine größere Blendegröße führt zu einer Berührung mit den Rauschpixeln ausserhalb des inneren Kreises, wobei es eine ungünstige Wirkung auf den Filter hat.

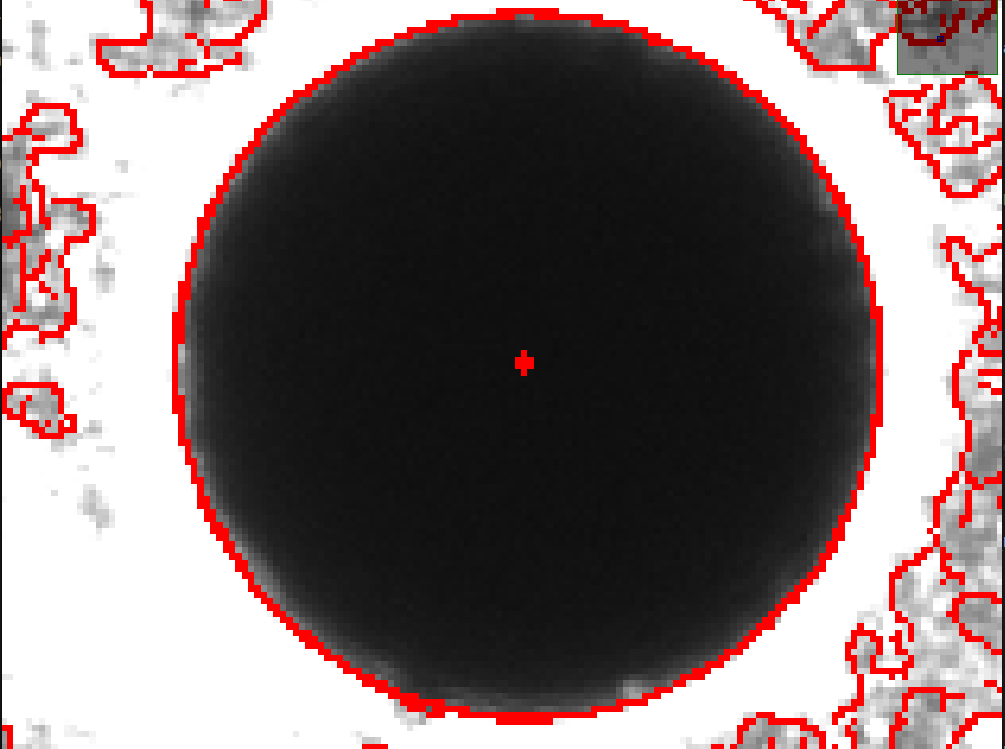
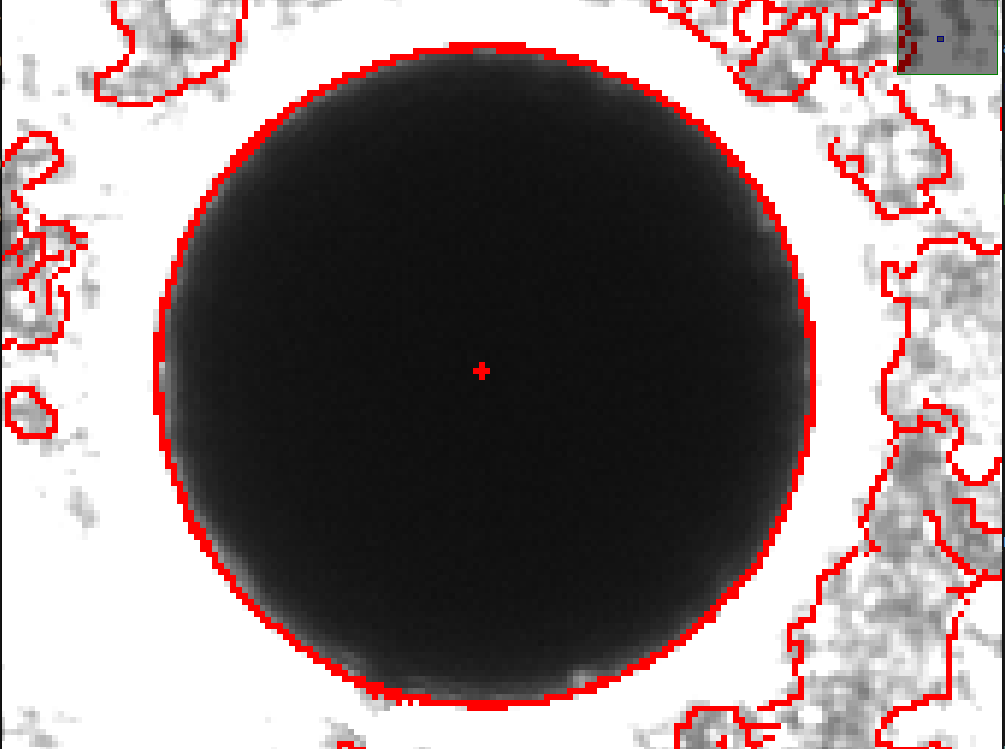
 

Abbildung 21: Detektion mit der Blendegröße 3, 5, 7, 9 von Median-Filter(von oben links nach rechts)

Es ist klar zu sehen, dass die spärlichen Rauchen außerhalb des inneren Kreises glättet werden und je größer der Blendegröße ist, desto wenigere Rauchsen extieren um die innere Kante herum. Diese Kante passt noch gut zu dem Kreis. Darüber hinaus wird der Detektionsfehler bei der lücke mit zunehmender Blendegröße immer kleiner.

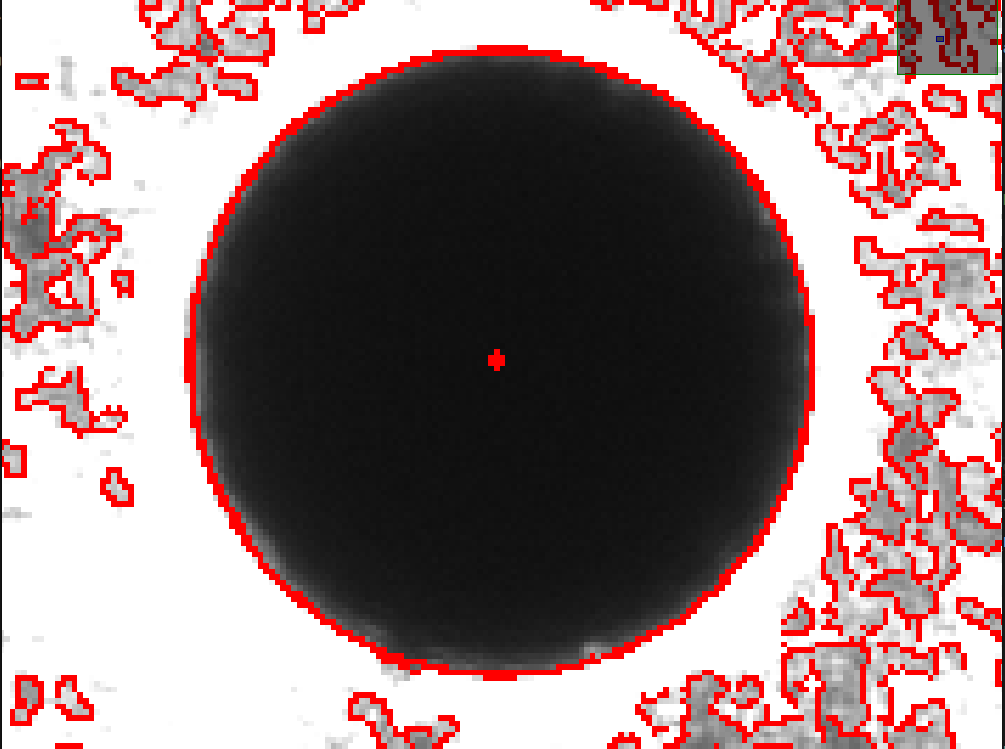
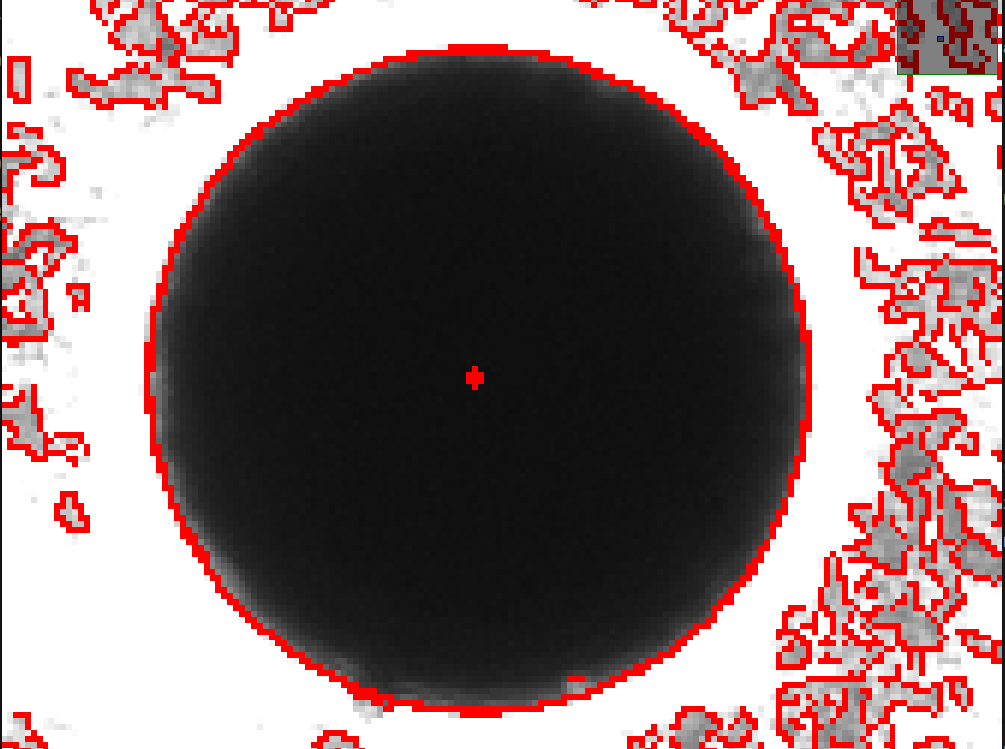
Wird der Canny-Edge-Detection-Algorithmus verwendet, ist die Rauschminderung typischerweise mit einem gaußschen Filter der erste Schritt.

Ein gaußscher Filter wird häufig verwendet, um Bildinhalte zu glätten oder weicher zu machen. Der Gauß-filter kann das Rauschen mit der Normalverteilung(Gauß-Verteilung) gut entfernen. In der Bildverarbeitung ist die Normalverteilung entsprechend zweidimensional, siehe Abbildung 22.



Abbildung 22: Zweidimensionale Normalverteilung[28]

Die Auswirkungen auf die Detektion mit gaußschem Filter sind wie in folgenden Bilden zu sehen.



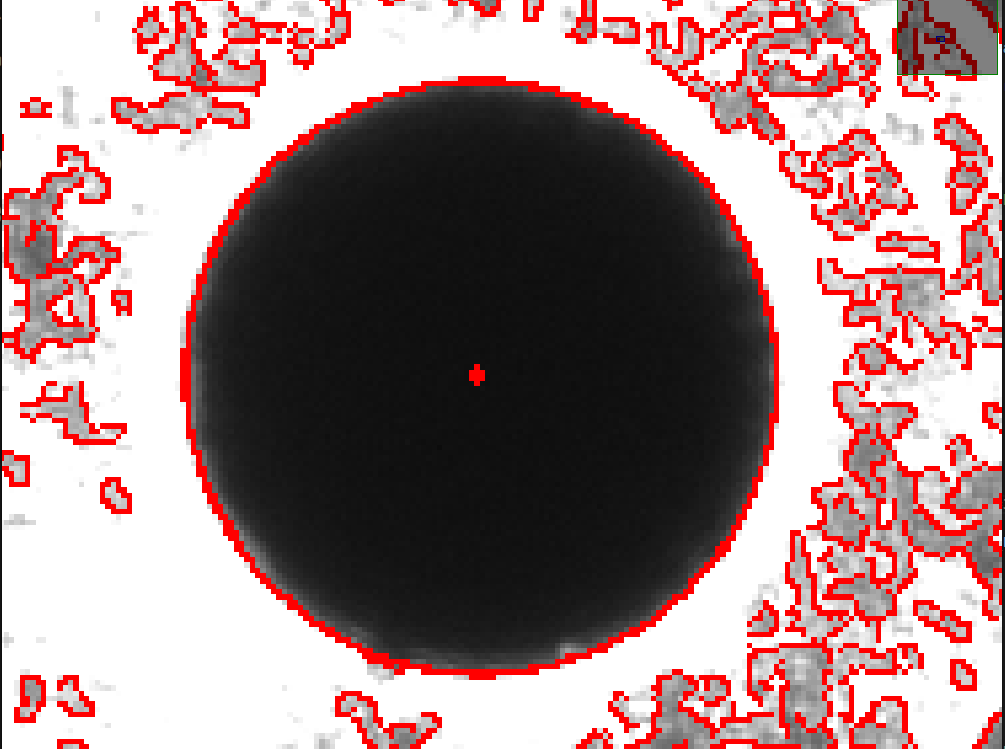
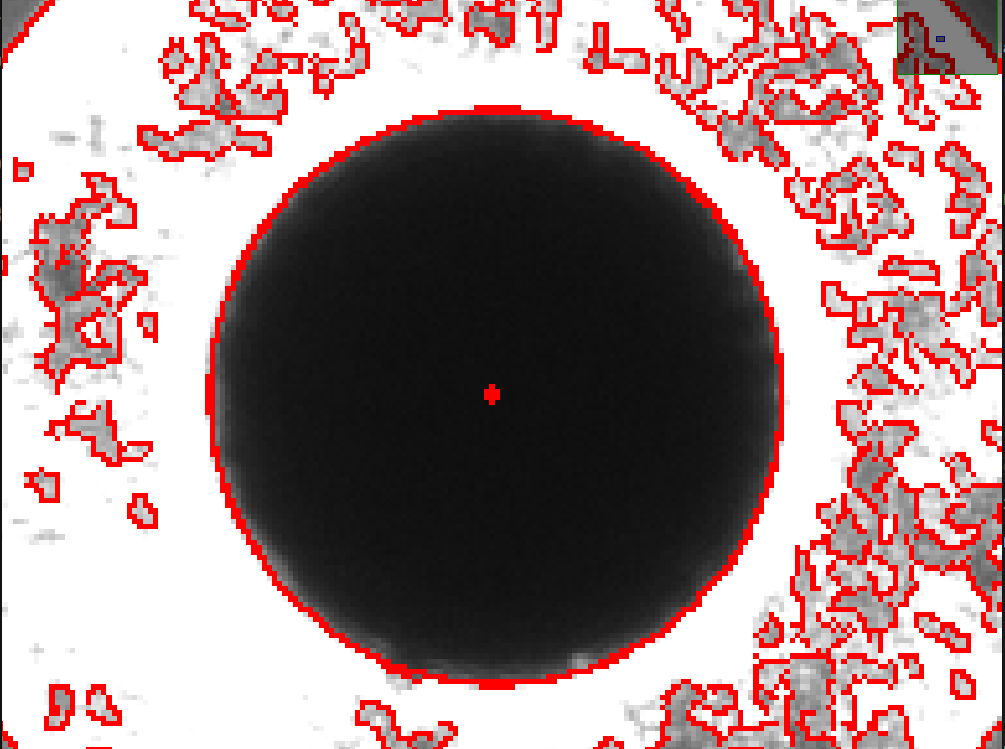


Abbildung 23: Detektion mit der Blendegröße 3, 5, 7, 9 von gaußschem Filter(von oben links nach rechts)

Es wird ebenso mit vier unterschiedlichen Blendegrößen versucht. Wie sie zeigen, passen sich die Kanten des inneren Keises gut trotz der veränderten Blendegrößen. Die Auswirkung auf die äußeren Rauschen ist schlecht. Die Minderung der Detektionsfehler bei der lücke ist nicht offensichtlich.

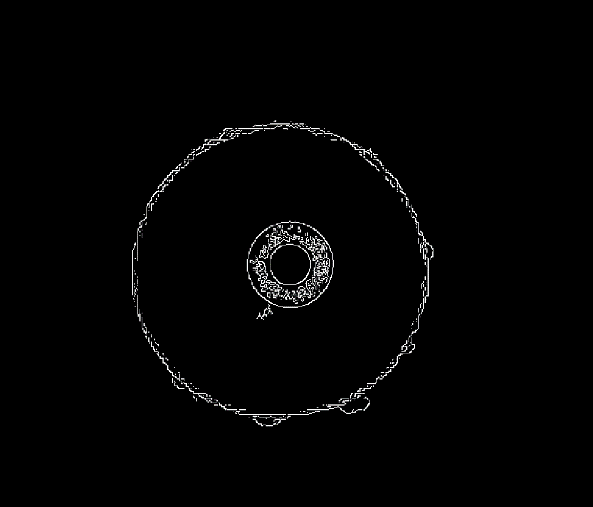
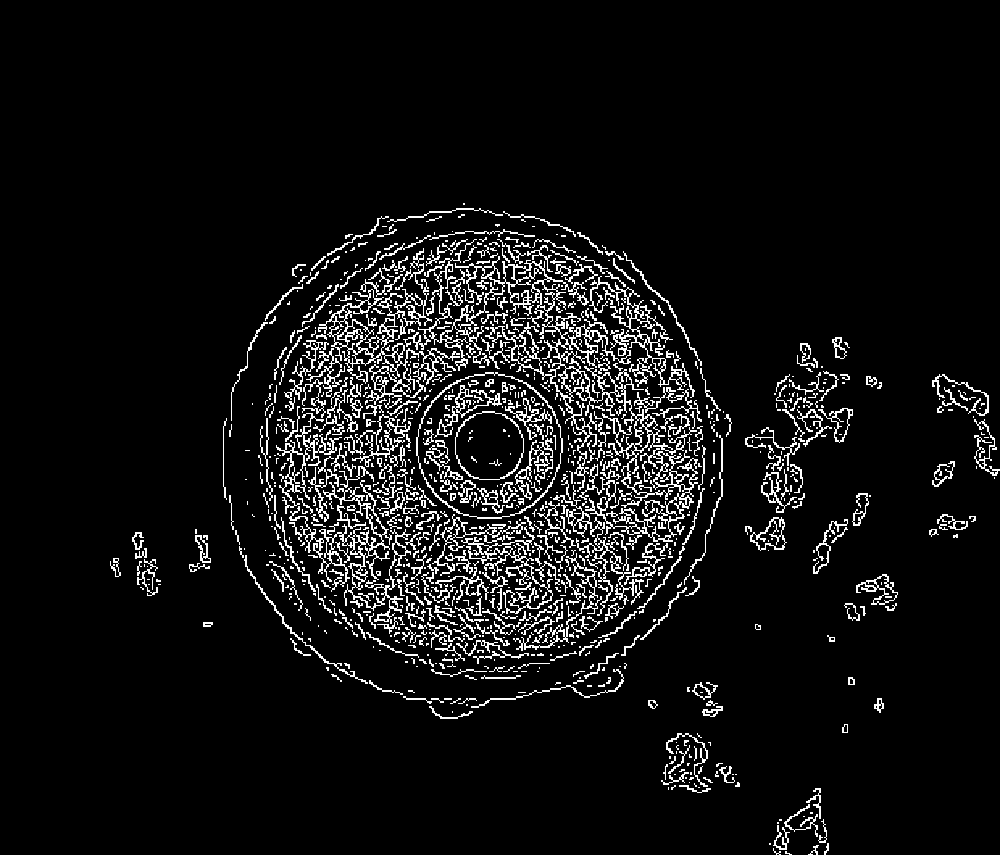
Wird nun die Auswirkungen dieser zwei Fliter verglichen, ist der Median-Filter in unserem Fall besser geeignet. Bei der Übereinstimmung der Kanten mit dem Kreis zeigen sie fast einen gleichen Effekt. Der Median-Filter zeigt vergleichweise eine bessere Auswirkung auf die Detektionsfehler bei der lücke. Außerdem hat der Median-Filter eine bessere Wirkung auf die äußeren Rauschen, womit ein geringerer Rechenaufwand kostet wird trotz der wenigen Beeinflussung dieses Rauchens auf die Detektion. Es wird deswegen der Median-Filter in dieser Aufgabe verwendet.

### 3.4.2 Kantendetektion mit Canny-Algorithmus

Die Canny()-Funktion hat insgesamt 6 Parameter: „image“, „edges“, „threshold1“, „threshold2“, „apertureSize“ und „L2gradient“. „image“ ist das Eingabeparamter. „edges“ ist das Ausgabeparameter, das ein Kantenbild ausgibt. ,L2gradient’ stellt es fest, ob eine präzisere Methode L2-Norm bei der Berechnung des Gradienten verwendet wird [29].

Die anderen drei Parameter sind die Parameter des Canny-Edge-Detection-Algorithmus, die die Performance dieses Algorithmus beschreiben. „threshold1“ und „threshold2“ sind die zwei Schwellwerte für das Hystereseverfahren. In unserem Fall sind diese zwei Werte nicht schwer zu finden. Wenn der größere Schwellwert auf 150 eingestellt, und der niedrigere auf einen empfohlenen Wert nämlich ein Drittel eingestellt ist das Ergebniss relativ gut.

„apertureSize“ stellt die Größe der Kernmatrix für den Sobel-Operator fest. Unten sind die drei Bilder, die mit unterschiedlichen Größen der Kernmatrix durchgeführt, dargestellt.



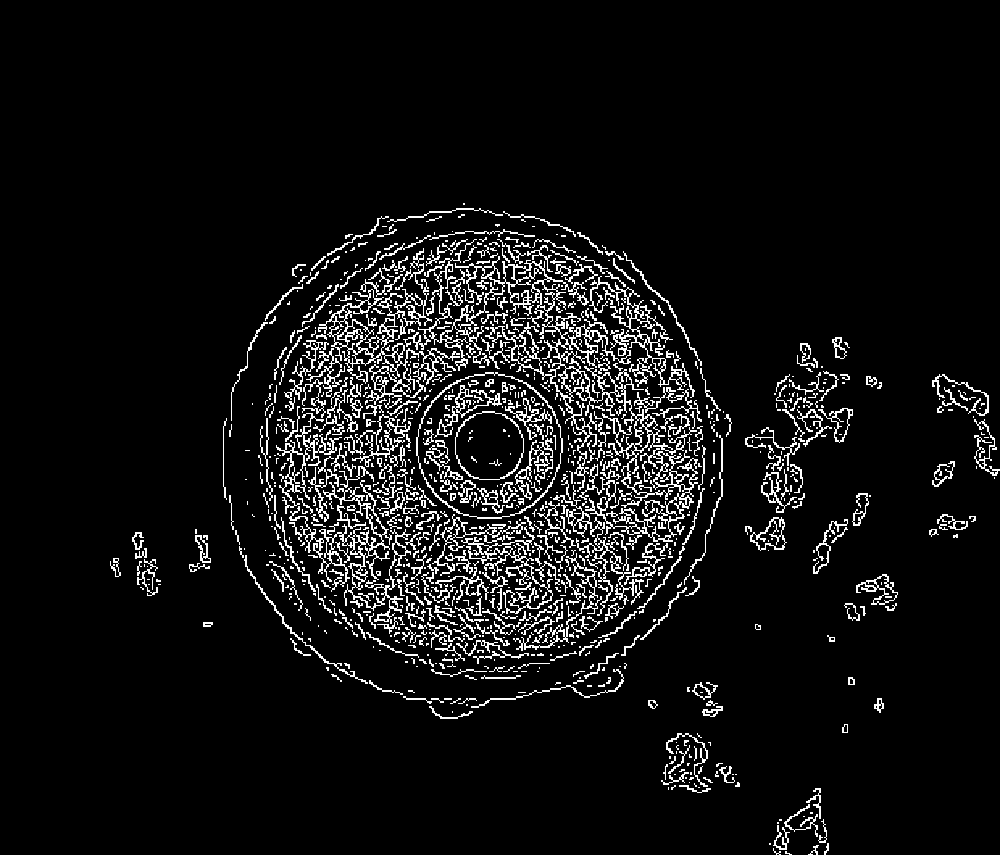


Abbildung 24: Canny-Detektion mit „apertureSize“ 3, 5, 7 (von oben links nach rechts)

Aus diesen Bilder ist es ersichtlish, dass die Wirkung von der Kerngröße mit 5 und 7 unbefriedigend sind. Konturen werden an vielen unnötigen Stellen erkannt, wodurch mehr Berechnungen und ein viel größer Speicherraum kostet werden. Im Versuch mit Kerngröß 3 wird der Breich außer dem Ring fast keine Konturen detektiert, was eine bessere Wirkung in dieser Aufgabe zeigt.

Die in unteren Unterkapitel durchgeführten Detektionen werden unter der Voraussetzung durchgeführt, dass dieser Parameter 3 beträgt.

### 3.4.3 Kreisdetektionsmethode

Um die für unsere Aufgabe best geeignete Methode auszuwählen, werden die Daten der detektierten Ellipsen des inneren Kreises mit den Funktionen fitEllipse(), fitEllipseAMS() und fitEllipseDirect() gesammelt, siehe Anhang I. Für jede Methode werden fünf Gruppen von Daten gesammelt. Da der Unterschied zwischen diesen Methoden in dem Bild kaum beobachtet werden kann, werden nur die schriftlichen Daten angezeigt. In der tatsächlichen Detektion werden oft zwei ähnlichen Ellipsen gleichzeitig detektiert werden. Dies kann von Abbildung 21 und Abbildung 24 entnommen werden. Hier für einen besseren Vergleich zwischen unterschiedlichen Methoden werden nur die Daten einer der Ellipsen angezeigt.

Wegen der ganz kleinen Differenz der detektierten Ellipse zwischen diesen drei Funktionen, ist enie weitere Forschung zu diesen Methoden von weniger Bedeutung. Es wird deshalb nur die Auswirkungen dieser drei Methoden betrachtet und die Diskussion ihrer Algorithmen wird in dieser Arbeit verzichtet.

Nun werden zuerst das zweite Element bzw. die Länge der Hauptachse und Nebenachse aller fünfzehn Gruppen Daten betrachtet. Die Differenz zwischen der Hauptachse und der Nebenachse ist jeweils ca. 0.201, 0.187 und 0.172 Pixeln. Bei einer Pixelgröße von 2.2 µm ist die Differenz jeweils 0.4422 µm, 0.4114 µm, 0.3784 µm. Obwohl fitEllipseDirect() ein besseres Ergebnis liefert, ist die Differenz zwischen diesen drei Methoden relativ klein und kann vernachlässigt werden. Wird der Mittelpunkt beobachtet, ist die Differenz unterschiedlicher Methoden ebenso ganz klein. Deswegen ist theoretisch jede dieser drei Methoden benutztbar.

Außerdem ist die Detektion des Kreises mit Ellipse gültig in dieser Aufgabe wegen der akzeptierbaren Abweichung zwischen der Hauptachse und Nebenachse.

Es wird dann der Trajektorienkreis aus der Bewegung des Grefersmittelpunkts mit diesen drei Methoden interpoliert. Die bekommenden Bilder inklusive den entsprechenden Daten sind wie folgend dargestellt. Die gezeigten vollen Kreise sind an den interpolierten Ellipsen angepasst, wobei der Radius die Hälfte aus Haupt- und Nebenachse ist. Für jede Methode werden drei Versuche durchgeführt.

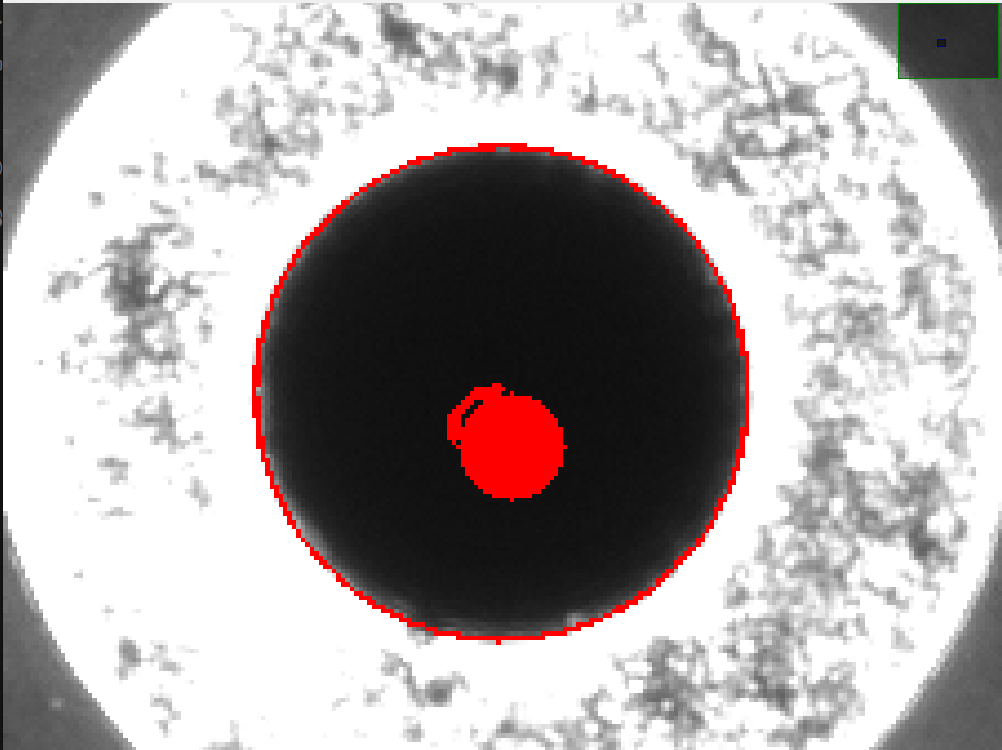
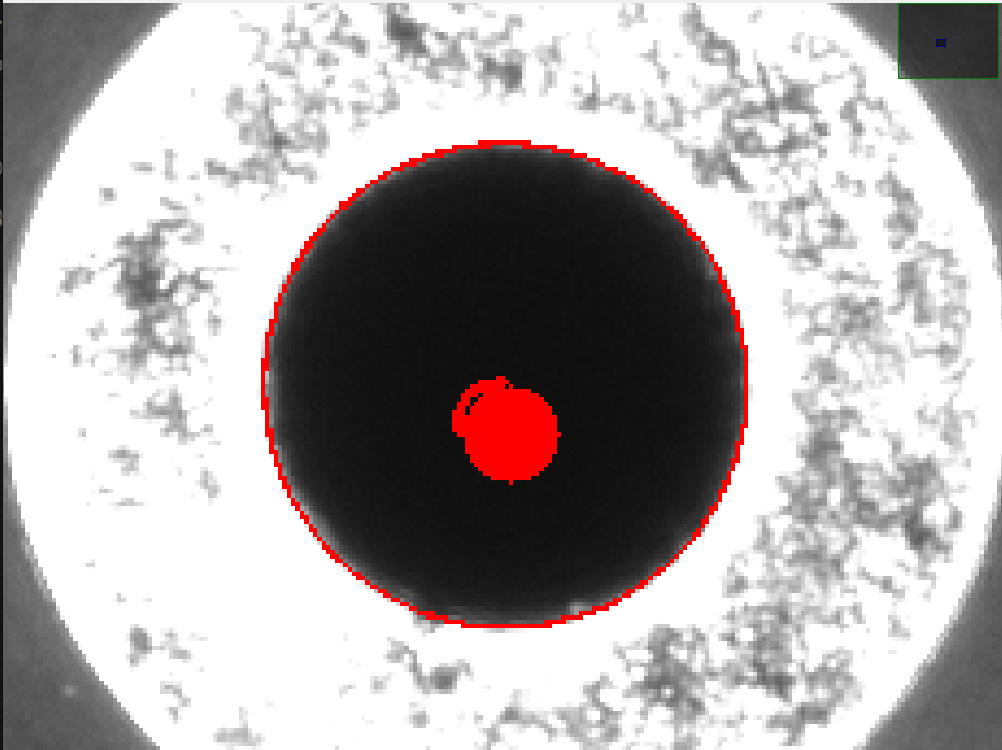
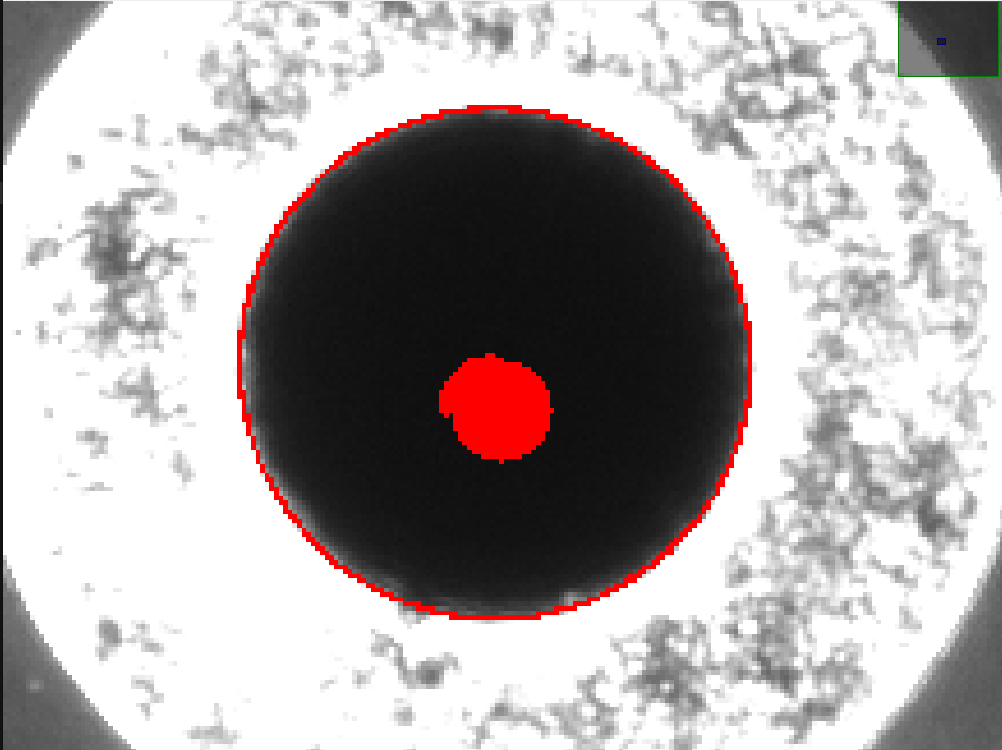
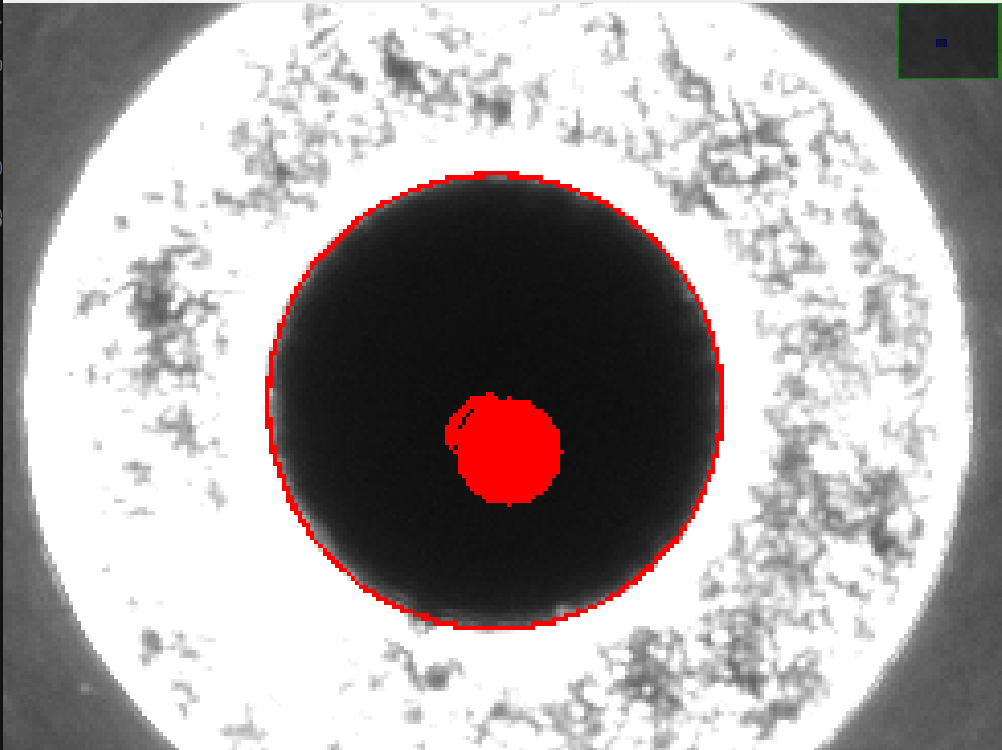


Abbildung 25: Interpolation mit fitEllipse() der drei Versuche (von oben links nach rechts)



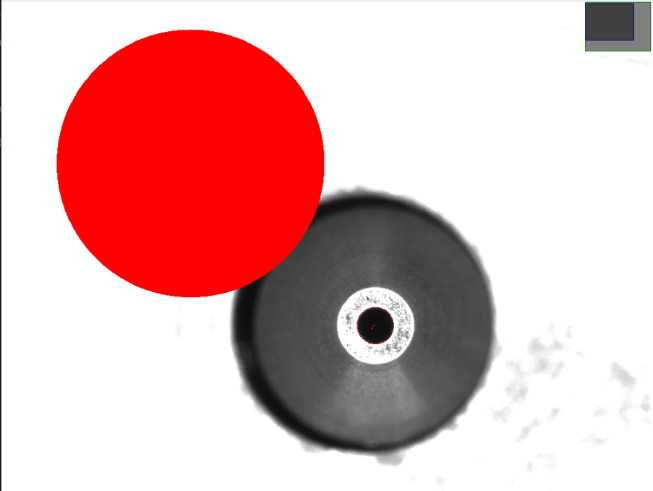
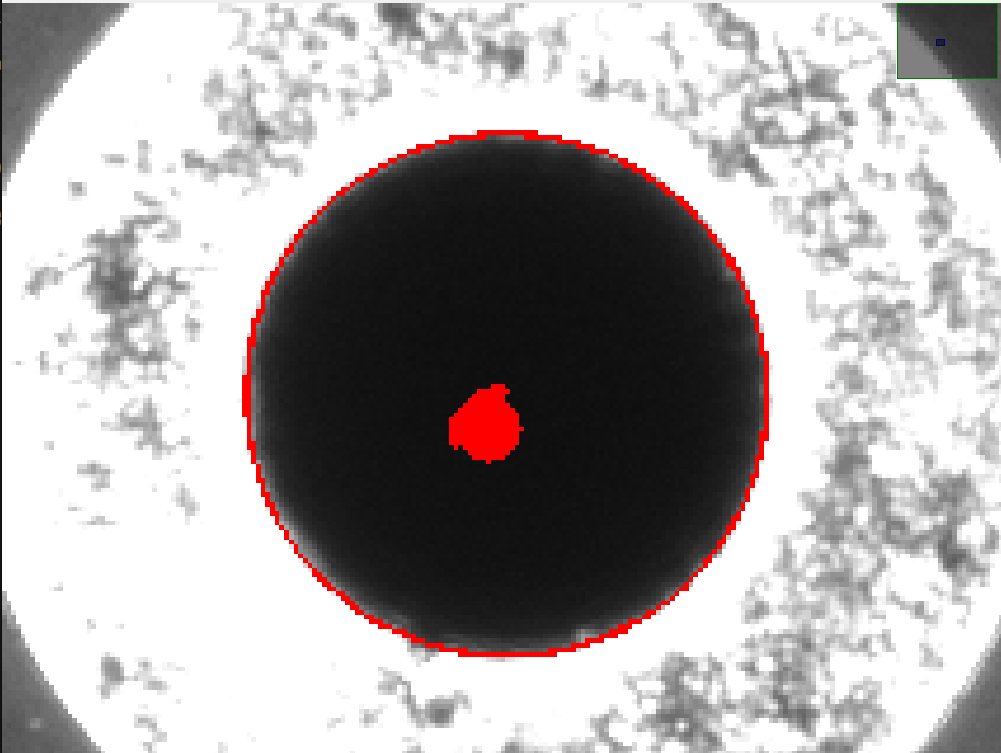
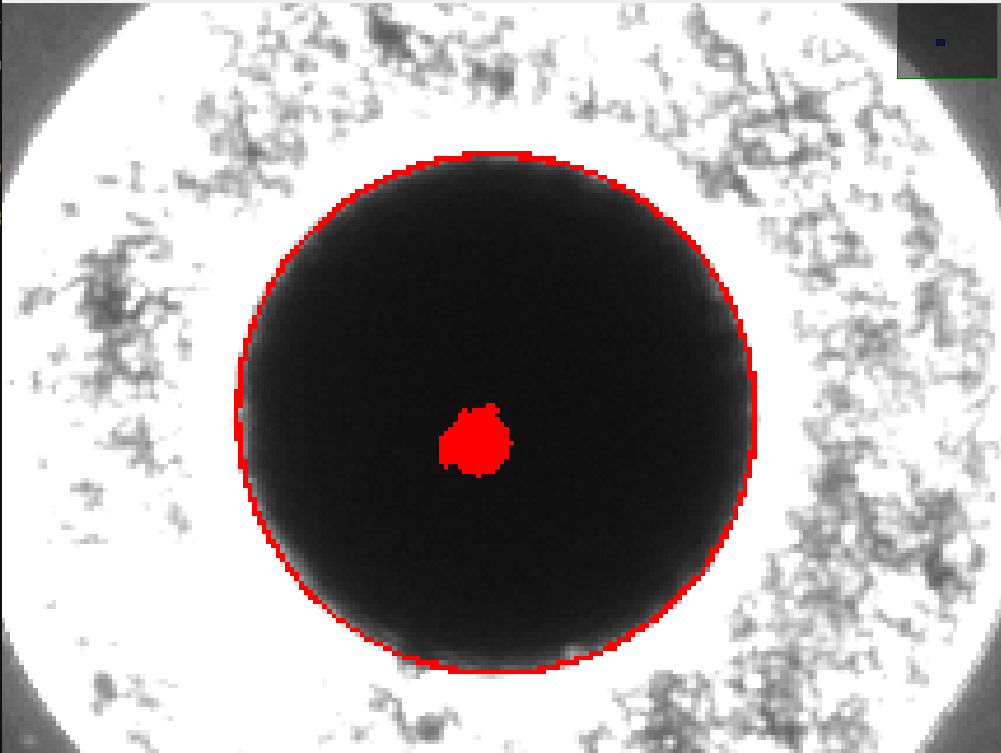


Abbildung 26: Interpolation mit fitEllipseAMS() der drei Versuche (von oben links nach rechts)



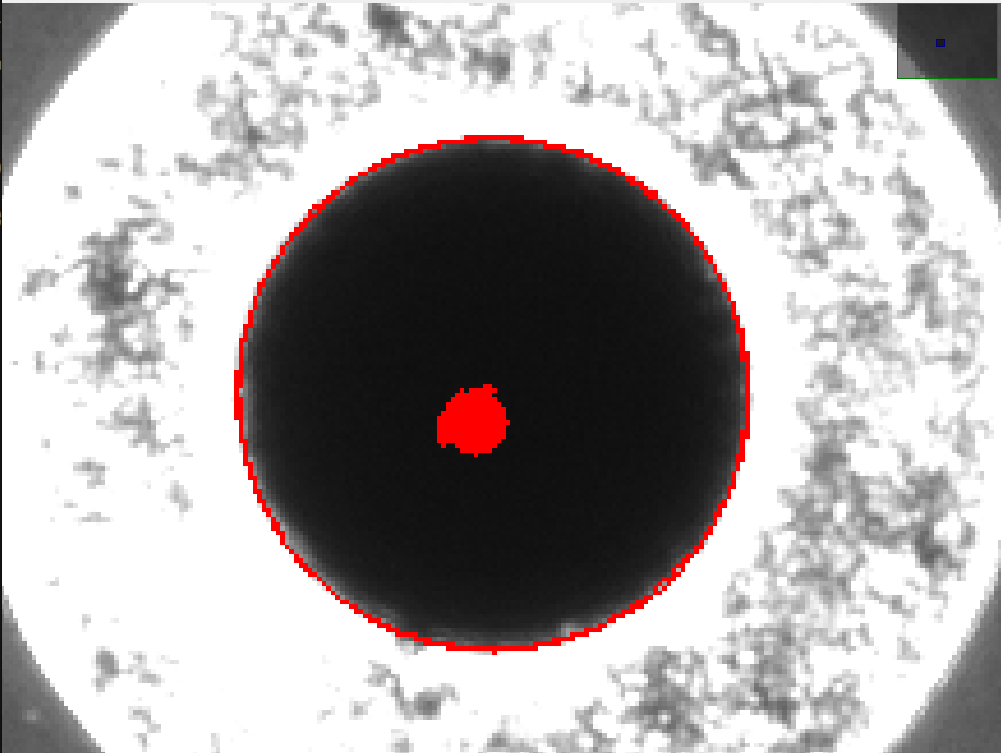


Abbildung 27: Interpolation mit fitEllipseDirect() der drei Versuche (von oben links nach rechts)

Durch die Beobachtung der obigen Bilder und Daten, ist es zu sehen, dass wegen der wenigen Datensätze ist die Differenz zwischen der Hauptachse und Nebenachse der interpolierten Ellipseist relativ groß. Die Methoden mit Ellipsendetektion liefern deswegen ein schlechtes Ergebnis.

Trotzdem liefert fitEllipseDirect() eine relativ stabiles Ergebnis. Es wird daher diese Methode verwendet für die vorherige Detektion des inneren Kreis.

Für die Interpolation der Bewegungsspur des Mittelpunkts ist diese Methode jedoch nicht geeignet. Es wird eine alternative Funktion hyperSVD() aus dem Modul circle\_fit verwendet. Es funktioniert gut, unabhängig davon, ob Datenpunkte entlang eines gesamten Kreises oder entlang eines kleinen Bogens abgetastet werden.

Folgend sind drei Versuche mit hyperSVD() inklusive ihre Daten.

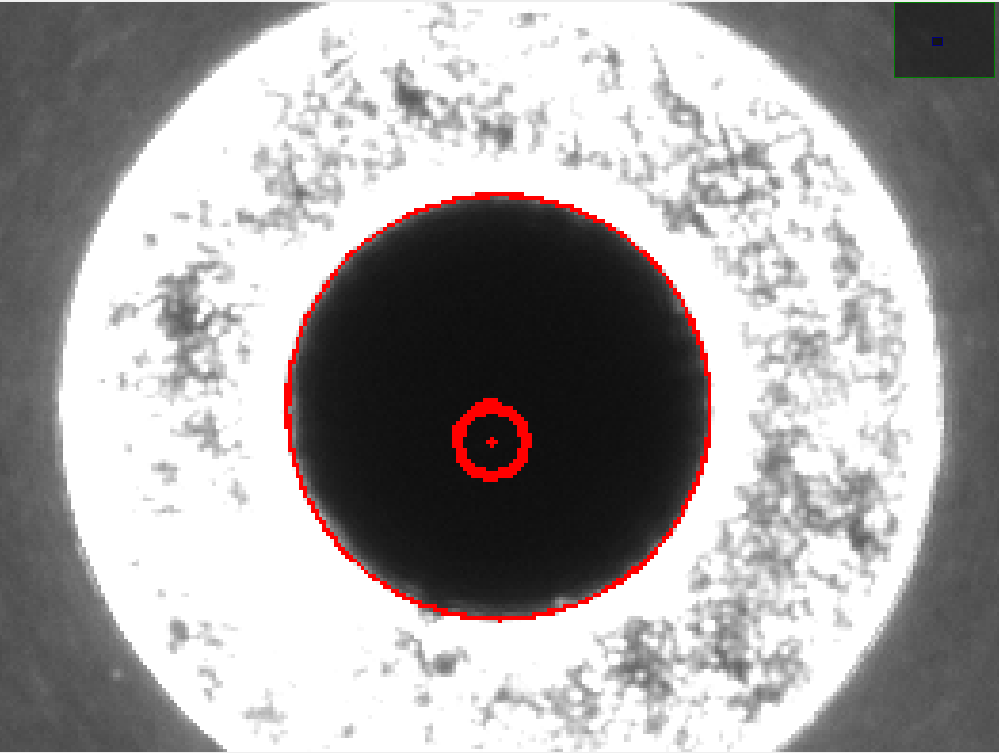


Abbildung 28: Interpolation mit hyperSVD() (Versuch 1)

Position des Greifers relativ zur roten Platte: (-20.19041827511974,2.4918963652570256) (μm)

Abweichung: 20.343611715215918 μm

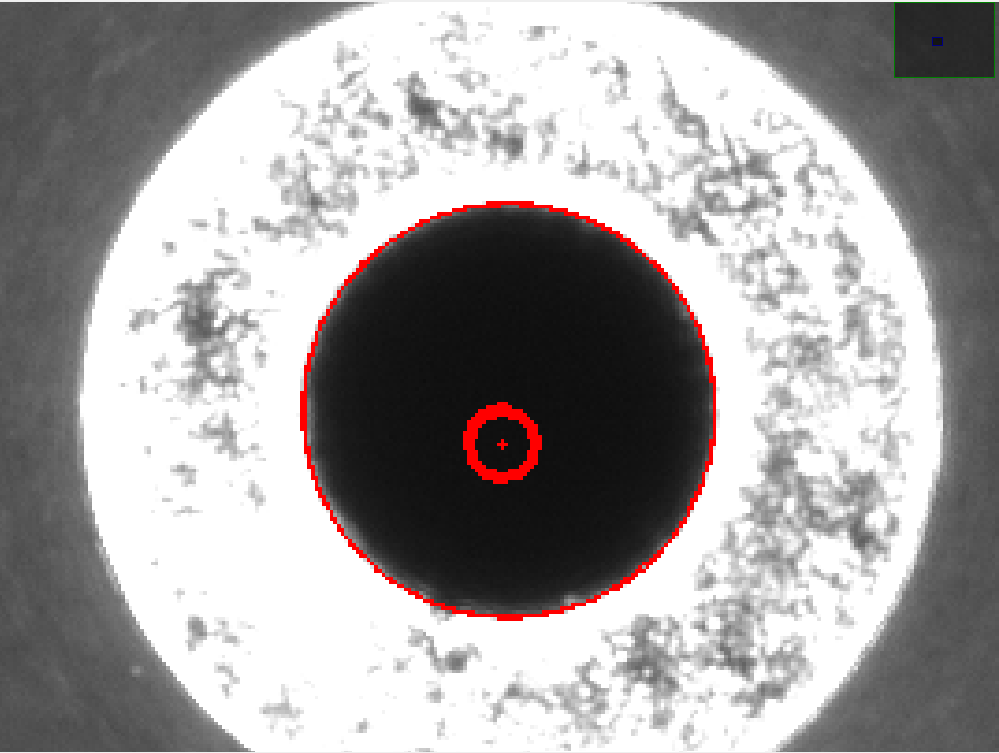


Abbildung 29: Interpolation mit hyperSVD() (Versuch 2)

Position des Greifers relativ zur roten Platte: (-20.18514448904889,2.4950189716784283) (μm)

Abweichung: 20.33876047631261 μm

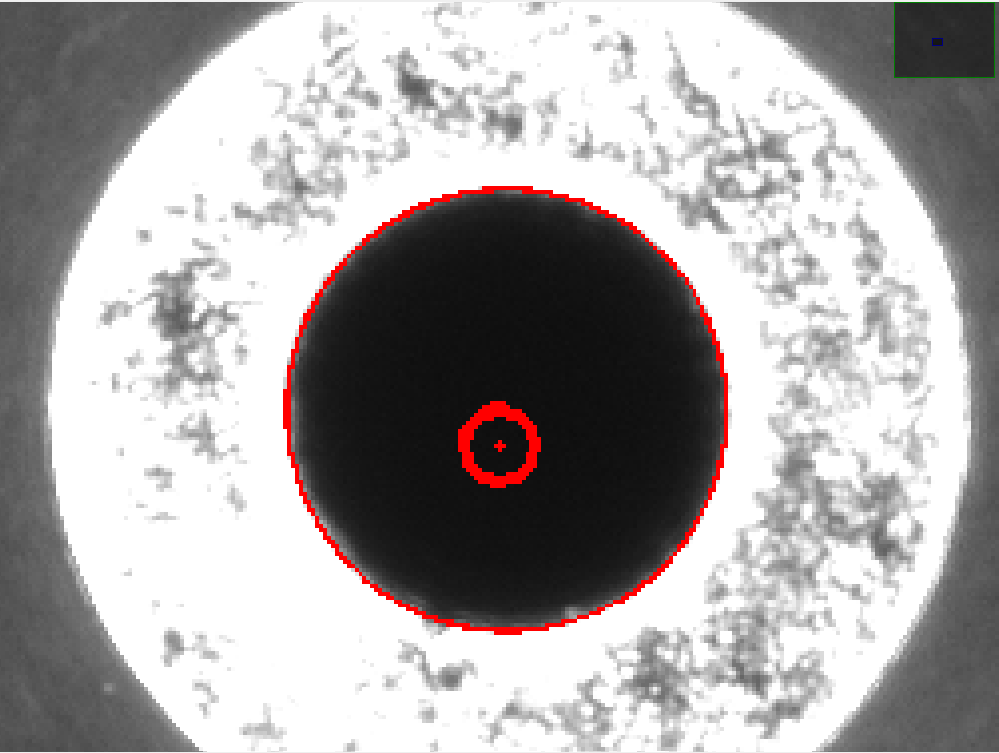


Abbildung 30: Interpolation mit hyperSVD() (Versuch 3)

Position des Greifers relativ zur roten Platte: (-20.275410691932667,2.4133427125112576) (μm)

Abweichung: 20.41853329146265 μm

Aus den drei interpolierten Mittelpunktskoordinaten ist es zu sehen, dass diese Methode bietet eine relativ gute Stabilität. Werden diese Bilder beobachtet, sind die interpolierten Kreise ebenso vernünftig.

### 3.4.4 Programmsdurchführung

Basierend auf den Funktionen sowie iheren Parameter aus vorherigen Unterkapiteln wird das Programm für die Detektion wie folgend dargestellt.

Nachdem das Rohbild aufgenommen wird, wird es zuerst ein Median-Filter implementiert. Dann wird es durch Canny() die Kanten des inneren Kreises detektiert. Mit findContours() werden die Daten des Kantenbildes aus Canny() extrahiert. In der extrahierten Konturen werden die möglichen erwünschten Ellipsen ausgewählt. Die Konditionen, nämlich die Länge der Haupt- und Nebenachse für die Auswahl der geeigneten Ellipse wurden durch eine manuelle Schätzung in dem detektierten Bild bekommen, die in unserem Fall ungefähr 110 Pixeln sind. Anschließend werden die Mittelpunkte der Ellipsen gesammelt für die spätere Interpolation.

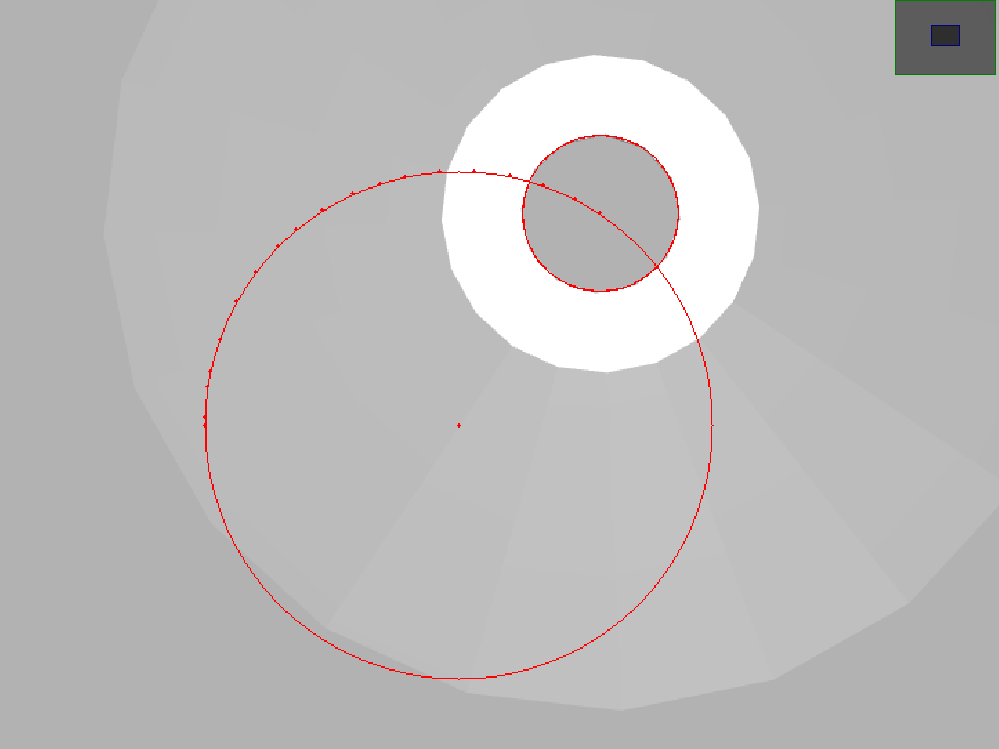
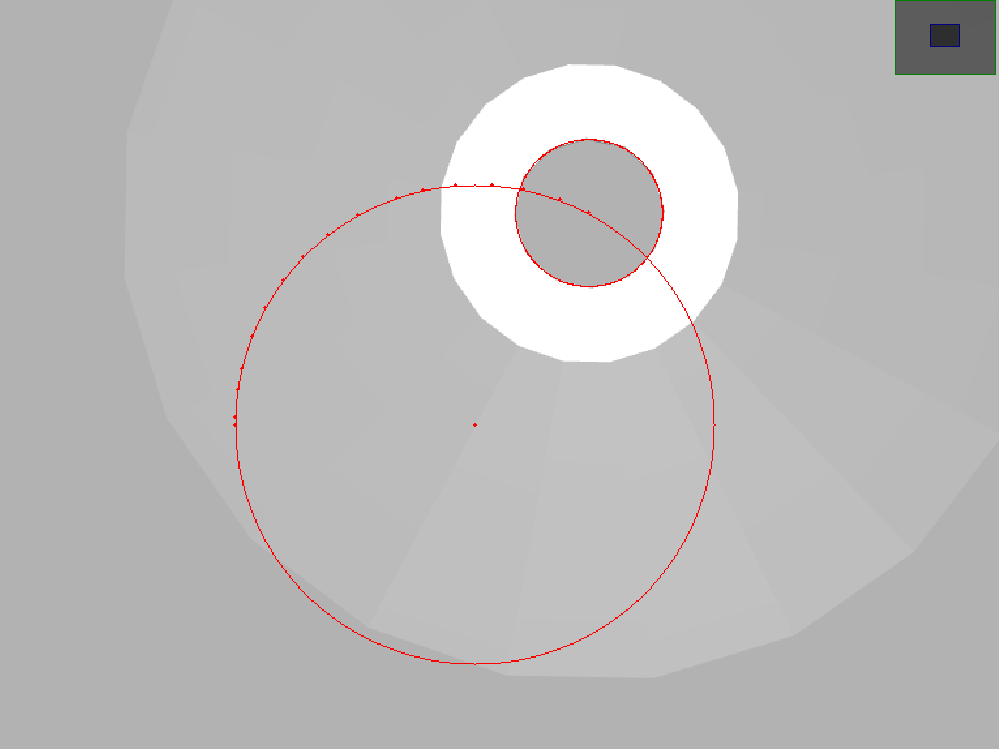
Da in der Simulationsumgebung ist die Pixelgröße unbekannt und es wird nur der Lochdurchmesser des Greiferkopfs gegeben, dessen Wert 500 µm beträgt, wird hier auch eine Rechnung der Pixelmenge des Radius durchgeführt. Es wird der Durchschnittswert der Haupt- und Nebenachse als den Radius des Kreises genommen. Die Pixelgröße in der Simulation beträgt damit 250 geteilt durch die Pixelgröße.

## 3.5 Genauigkeitsanalyse und Fehleranalyse

In diesem Kapitel wird eine Analyse zur Genauigkeit der verwendeten Kalibriermethode sowie einige möglichen Fehlerursachen durchgeführt.

### 3.5.1 genauigkeit

Die Analyse der erreichten Genauigkeit wird in der Simulationsumgebung durchgeführt. Zuerst wird nur die Abweichung in einer Achse betrachtet. Hier wird eine Abweichung in der y-Achse auf 800 µm eingestellt. Die Kalibeirungsergebnisse von den drei Versuche sind wie folgend dargestellt.



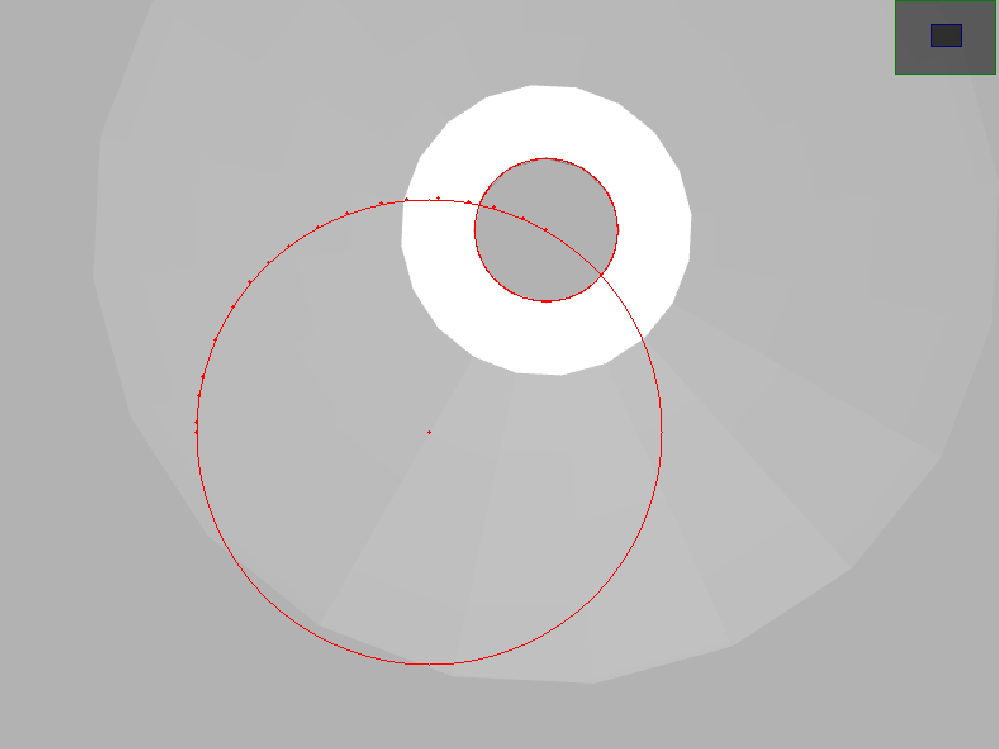
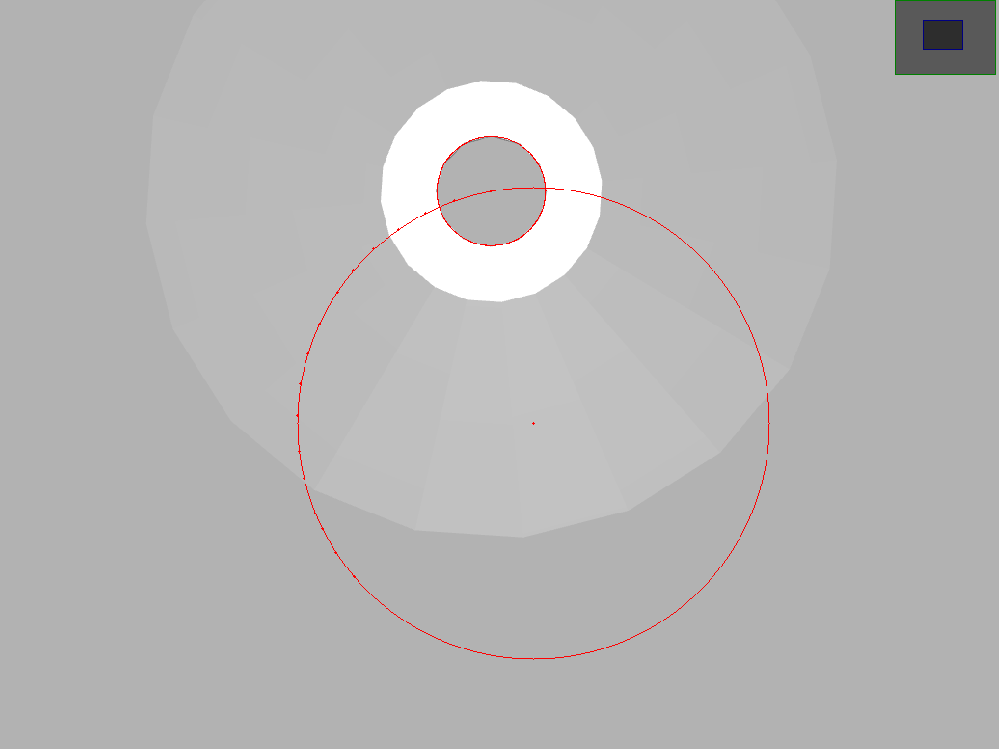
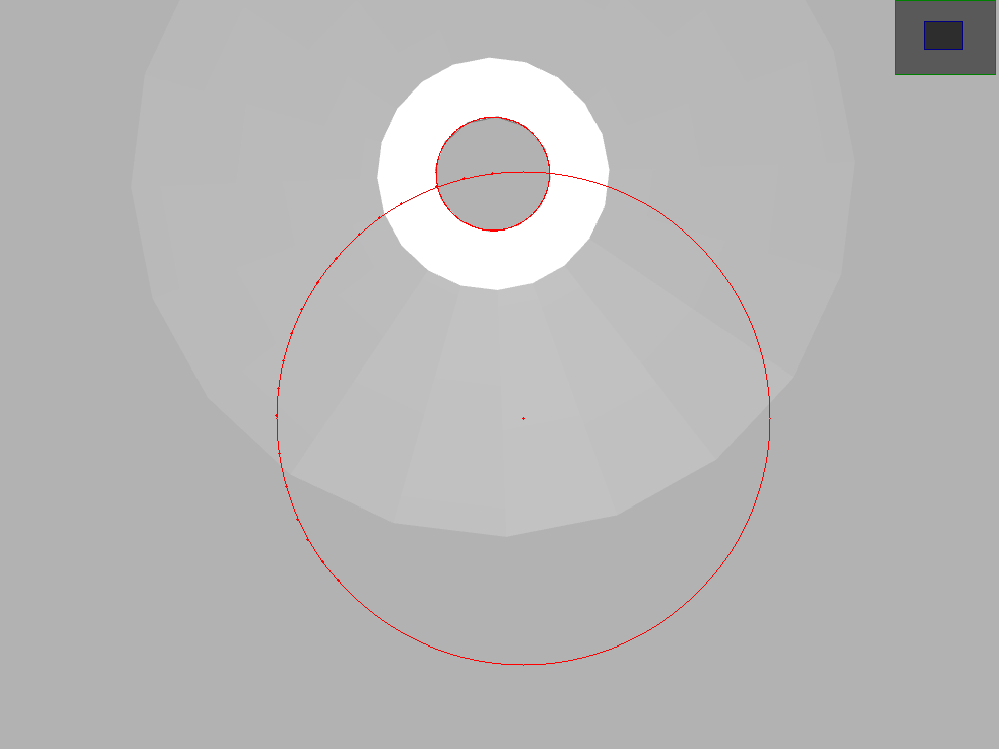


Abbildung 31: Kalibeirungsergebnisse der drei Versuche mit Abweichung in y-Achse (von oben links nach rechts)

Werden die eingestellte Abweichung durch die detektierte Abweichung dividiert, sind die Genauigkeit der drei Versuche jeweils ca. 97.723 %, 97.716 % und 97.728 %, wobei eine durchschnittliche Genauigkeit ca. 97.722% erreicht.

Es werden dann eine Abweichung von 700 µm in der x-Achse und eine Abweichung von 800 µm in der y-Achse eingestellt. Die drei Ergebnisse sind unten dargestellt.



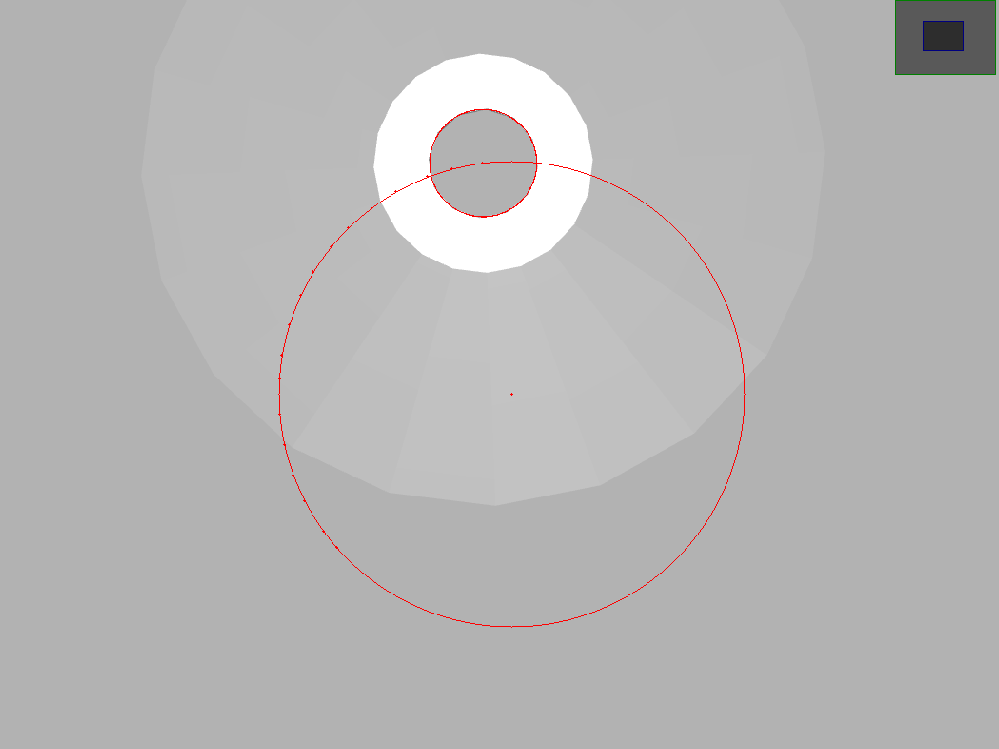


Abbildung 32: Kalibeirungsergebnisse der drei Versuche mit Abweichung in x-Achse und y-Achse (von oben links nach rechts)

Die erreichte Genauigkeit des jeweiligen Versuchs ist ca. 97.757 %, 97.731 %, 97.729 %. Die durchschnittliche Genauigkeit beträgt 97.739 %.

Werden noch einmal ein Durchschnittswert von diesen zwei durchschnittlichen Genauigkeit ausgerechnet, beträgt eine endliche Genauigkeit 97.7305 %. Dieser Arbeit hält dies für ein vernünftiges Ergebnis. Trotz der gewissen Fehlerrate hat die in dieser Arbeit verwendete Kalibriermethode eine positive Wirkung auf dem Real2Sim-Gap erzielt.

Da in der Simulationsumgebung wird die Pixelgröße durch die Schnittswerts manuell ausgerechnet, kann die Genauigkeit der Detektion der realen Anlage etwas höher sein.

### 3.5.2 Fehleranalyse

In Kombination mit dem vorherigen Prozess der Detektion fasst diese Arbeit enige möglichen Fehler zusammen.

In dieser Arbeit wird die Funktion fitEllipseDirect() in OpenCV verwendet, um den Kreis zu detektieren. In obigen Unterkapiteln wurde es schon nachgewiesen, dass diese Methode vernünftig ist wegen ihrer guten Performance. Trotzdem ist dieser Fehler unvermeidlich.

Der in dem Unterkapitel 4.3 erwähnte Detektionsfehler an der Lücke sowie die Unebenheit des inneren Kreises können auch nicht vernachlässigt werden. Außerdem aufgrund der Möglichkeit, zwei oder mehrere Kreise gleichzeitig zu detektieren, wird der detektierte Mittelpunkt durch die Durchschnittswerte bestimmt. Dies wird als eine Fehlerursache gezählt.

Diese Gründen hat auch den Fehler der auf diesen Mittelpunkte basierte Interpolation der kreisförmigen Bewegungsspur zur Folge.

Einige systematische Fehler wie z. B. die Toleranz von den Controller sind zu vernachlässigen wegen ihrer ganz kleinen Beeinflussungen.

## 3.6 Kalibrierung des Modells

Nachdem die Detektion sowie die Berechung des Positionsfehlers fertig sind, besteht ein weiterer unerlässlicher Schritt darin, das Simulationsmodell anzupassen. Dann kann die kalibrierungsaufgabe vollständig beendet werden. Dies erfolgt durch die automatische Anpassung der „yaml“ Datei. Unter dem Arbeitsordner befindet sich eine Datei „pm\_robot\_joint\_calibration.yaml“. Diese Datei gibt an, wie groß die ursprüngliche Abweichungen der jeweiligen Bauteile sind. Die Beschreibung jedes Bauteil enthält die Abweichung in x-, y- und z-Achse sowie die Rotationsabweichung um diese drei Achsen. Im unseren Fall sind die zu anpassenden Teile die Abweichung in der x- und y-Achse.

Um diese „yaml“ Datei zu ändern wird die Class YAML() im Python-Modul ruamel.yaml verwendet. Das Programm ist wie folgend dargestellt.

Es wird zuerst die Class YAML() instanziiert. Dann wird die Datei geöffnet mit der in Python integrierten Funktion open() und geladen mit der Methode load() in der Class YAML(), sodass die Inhalt der Datei gändert werden kann. Danach folgt es die Speicherung der Änderung mit der Methode dump() [30]. Schließlich wird eine Message „Calibration successful!“ ausgegeben, um zu zeigen, dass die Anpassung der Datei erfolgreich ist.

Zu beachten ist, eine Koordinatentransformation kann hier nicht vernachlässigt werden. Durch einen Vergleich der Bildkoordinate mit der realen Bewegungsrichtung des Greifers ist es nicht schwer zu finden, dass eine reale Bewegung in der positiven y-Richtung zu einer Änderung der negativen y-Richtung in der Bildkoordinate führt. Dewegen muss ein Minuszeichen hinzugefügt werden bei der Anpassung des Werts in der y-Richtung.

Da die in der Datei verwendete Einheit Mikrometer ist, eine Einheitenumrechnung nicht erforderlich.

Ein vollständiger Kalibrierungsprozess ist bis hier fertig.

# 4. Zusammenfassng und Ausblick

Mithilfe von ROS 2 und OpenCV wird es in dieser Arbeit das Real2Sim-Gap einer Präzisionsmontageanlage verkleinert. Das Real2Sim-Gap resultiert aus der Abweichung bei der Montage des Vakuumgreifers.

Dazu wurden zuerst die Positionierung und eine anschließende Rotation des Vakuumreifers durchgeführt. Mithilfe der OpenCV wurde es während der Rotation eine Detektion des kreisförmigen Greifers erreicht. Das Steuern des Greifers wurde durch die „Action“-Kommunikation erreicht. In der Detektion wurde zuerst ein Median-Filter verwendet um den Bild vorzubearbeiten. Bei der Kantendetektion wurde der Canny-Algorithmus verwendet wegen seiner hohen Genauigkeit. Die Locherkennung basiert auf der Ellipsendetektion und anhand ihrer Mittelpunkte wurden die Montageabweichung des Vakuumgreifers bzw. das Real2Sim-Gap berechnet. Durch die Anpassung bestimmter Datei wurde die Kalibrierung des Modells abgeschlossen. Schließlich wurde es in Kombination mit der Kalibrierungsmethode eine Analyse der möglichen Fehlerursachen durchgeführt.

Die erreichte Genauigkeit in der Simulationsumgebung beträgt 97.7305 %.

Die in dieser Arbeit verwendete Methode reduziert den Real2Sim-Gap bis zu einem gewissen Grad. Basierend auf den bekannten Fehlern wird es hier einige möglichen Verbesserungen gegeben, um den Detektionsfehler weiter zu reduzieren.

In der Vorbearbeitung des Bildes wurde es in dieser Arbeit nur die Auswirkung einzelnes Filters besprochen. Für ein besseres Ergenis kann mehrere Filter kombiniert werden. Die Lücke an dem Vakuumgreifer muss in der Bildvorbearbeitung ausgliechen werden oder durch eine andere Methode kalibriert werden. Es kann versucht werden, das Loch bzw. den inneren Kreis mit anderen Kreiserkennungsalgorithmen zu detektieren.

In dieser Arbeit ist der Kalibrierungsprozess eine einmalige Kalibrierung. Es kann so konzipiert, dass die Kalibrierung mehrmalig und mit unterschiedlichen Rotationswinkel durchgeführt wird um ein präziseres Ergebnis zu bekommen.

Diese möglichen Verbesserungen basieren nur auf der in dieser Arbeit verwendeten Kalibrierungsvorgehensweise.

Außerdem bei dem Aspekt Programmierung ist die Logik noch zu verbessern.

# 5. Literaturverzeichnis

1. Gregory, Hollows und James, Nicholas, 2023. Die Vorteile der Telezentrie. [online]. Mainz: Edmund Optics Inc. vertreten durch Edmund Optics GmbH [Zugriff am: 24.07.2023]. Verfügbar unter: <https://www.edmundoptics.de/knowledge-center/application-notes/imaging/advantages-of-telecentricity>
2. ANNIKA Raatz, 2021. Präzisionsmontage. Messtechnik. Vorlesungsfolien. Seite 63
3. Khaled,Elmadawi, 2021. Real2Sim: Realistic simulation environments. In: discourse.ros.org [online]. 04.2021 [Zugriff am: 18.02.2023]. Verfügbar unter: <https://discourse.ros.org/t/real2sim-realistic-simulation-environments/19857>
4. Open Robotics, 2021. The ROS Ecosystem [online]. California: Open Robotics [Zugriff am: 19.02.2023]. Verfügbar unter: <https://www.ros.org/blog/ecosystem>
5. Habib, Oladepo, 2018. Nodes [online]. California: Open Robotics. 04.12.2018 [Zugriff am: 19.04.2023]. Verfügbar unter: http://wiki.ros.org/Nodes
6. Steve, Macenski, Foote, Tully, Gerkey, Brian, Lalancette, Chris, Woodall, William, 2022. Robot Operating System 2: Design, Architecture, and Uses In The Wild [online]. 14.11.2022 [Zugriff am: 11.03.2023]. Verfügbar unter: <https://arxiv.org/pdf/2211.07752.pdf>
7. Open Robotics, 2023. Understanding nodes [online]. California: Open Robotics [Zugriff am: 19.03.2023]. Verfügbar unter: <https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html>
8. Open Robotics, 2023. About ROS 2 interfaces [online]. California: Open Robotics [Zugriff am: 19.04.2023]. Verfügbar unter: <https://docs.ros.org/en/foxy/Concepts/About-ROS-Interfaces.html>
9. Wikipedia, 2023. OpenCV [online]. San Francisco: Wikipedia [Zugriff am: 30.07.2023]. Verfügbar unter: <https://de.wikipedia.org/wiki/OpenCV>
10. doxygen, 2023. Introduction [online]. California: Doxygen [Zugriff am: 15.06.2023]. Verfügbar unter: <https://docs.opencv.org/4.7.0/d1/dfb/intro.html>
11. doxygen, 2023. Sobel Derivatives [online]. California: Doxygen [Zugriff am: 10.06.2023]. Verfügbar unter: <https://docs.opencv.org/4.x/d2/d2c/tutorial_sobel_derivatives.html>
12. GitHub, 2023. Image Processing [online]. California: GitHub [Zugriff am: 16.06.2023]. Verfügbar unter: <https://github.com/meredithheskethfortescue/MIREVI_FMA_IntSys_SS19>
13. Linkedin, 2023. What are the advantages and disadvantages of using Sobel, Canny, and Laplacian filters for edge detection? [online]. California: Linkedin [Zugriff am: 26.07.2023]. Verfügbar unter: <https://www.linkedin.com/advice/1/what-advantages-disadvantages-using-sobel-canny>
14. doxygen, 2023. Laplace Operator [online]. California: Doxygen [Zugriff am: 10.06.2023]. Verfügbar unter: <https://docs.opencv.org/4.x/d5/db5/tutorial_laplace_operator.html>
15. BIG VISION LLC, 2023. Edge Detection Using OpenCV [online]. San Diego: BIG VISION LLC [Zugriff am: 18.06.2023]. Verfügbar unter: <https://learnopencv.com/edge-detection-using-opencv>
16. IndianTechWarrior, 2023. Canny Edge Detection for Image Processing [online]. Gurugram Haryana: IndianTechWarrior [Zugriff am: 21.06.2023]. Verfügbar unter: <https://indiantechwarrior.com/canny-edge-detection-for-image-processing>
17. Wikipedia, 2023. Image gradient [online]. San Francisco: Wikipedia [Zugriff am: 05.06.2023]. Verfügbar unter: <https://en.wikipedia.org/wiki/Image_gradient>
18. doxygen, 2023. Canny Edge Detection [online]. California: Doxygen [Zugriff am: 6.06.2023]. Verfügbar unter: <https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html>
19. Wikipedia, 2023. Canny edge detector [online]. San Francisco: Wikipedia [Zugriff am: 06.06.2023]. Verfügbar unter: <https://en.wikipedia.org/wiki/Canny_edge_detector>
20. Wikipedia, 2023. Hysteresis [online]. San Francisco: Wikipedia [Zugriff am: 15.06.2023]. Verfügbar unter: <https://en.wikipedia.org/wiki/Hysteresis>
21. doxygen, 2023. Canny Edge Detection [online]. California: Doxygen [Zugriff am: 11.06.2023]. Verfügbar unter: <https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html>
22. doxygen, 2023. Contours : Getting Started [online]. California: Doxygen [Zugriff am: 01.06.2023]. Verfügbar unter: <https://docs.opencv.org/4.7.0/d4/d73/tutorial_py_contours_begin.html>
23. WordPress, 2020. Finding and Drawing Contours [online]. WordPress [Zugriff am: 15.06.2023]. Verfügbar unter: <https://cvexplained.wordpress.com/2020/06/03/finding-and-drawing-contours>
24. Open Robotics, 2020. Raw Message Definition [online]. California: Open Robotics [Zugriff am: 09.06.2023]. Verfügbar unter: <https://docs.ros2.org/foxy/api/trajectory_msgs/msg/JointTrajectoryPoint.html>
25. Open Robotics, 2023. About ROS 2 interfaces [online]. California: Open Robotics [Zugriff am: 09.05.2023]. Verfügbar unter: <https://docs.ros.org/en/foxy/Concepts/About-ROS-Interfaces.html>
26. Open Robotics, 2022. Action Definition [online]. California: Open Robotics [Zugriff am: 10.06.2023]. Verfügbar unter: <http://docs.ros.org/en/noetic/api/control_msgs/html/action/FollowJointTrajectory.html>
27. R. Fisher, S. Perkins, A. Walker und E. Wolfart, 2003. Median Filter [online]. [Zugriff am: 12.06.2023]. Verfügbar unter: <https://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm>
28. PixelsTech.net, 2020. Understanding nodes [online]. PixelsTech.net [Zugriff am: 19.03.2023]. Verfügbar unter: <https://www.pixelstech.net/article/1353768112-gaussian-blur-algorithm>
29. doxygen, 2023. Feature Detection [online]. California: Doxygen [Zugriff am: 10.06.2023]. Verfügbar unter: <https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga04723e007ed888ddf11d9ba04e2232de>
30. Python Land, 2023. Python YAML: How to Load, Read, and Write YAML [online]. Amsterdam: Python Land [Zugriff am: 19.07.2023]. Verfügbar unter: <https://python.land/data-processing/python-yaml#google_vignette>

**Anhang**

**I Detektionsdaten mit drei Algorithmen zur Ellipsendetektion**

fitEllipse():

((1124.317626953125, 1039.4912109375), (110.39801025390625, 110.79464721679688), 153.5677490234375)

-------------------------------------

((1124.322509765625, 1039.49072265625), (110.39603424072266, 110.82402801513672), 155.80288696289062)

-------------------------------------

((1124.3331298828125, 1039.4881591796875), (110.41492462158203, 110.76931762695312), 154.2998504638672)

-------------------------------------

((1124.33349609375, 1039.4561767578125), (110.39838409423828, 110.84590148925781), 154.88470458984375)

-------------------------------------

((1124.313720703125, 1039.4739990234375), (110.43585968017578, 110.82440185546875), 154.7988739013672)

fitEllipseAMS():

((1124.3155517578125, 1039.499755859375), (110.37834930419922, 110.76835632324219), 147.96676635742188)

-------------------------------------

((1124.313232421875, 1039.5072021484375), (110.36090850830078, 110.78636932373047), 151.14756774902344)

-------------------------------------

((1124.3157958984375, 1039.5029296875), (110.38514709472656, 110.74469757080078), 148.13067626953125)

-------------------------------------

((1124.32080078125, 1039.503662109375), (110.43389892578125, 110.75997161865234), 153.09568786621094)

-------------------------------------

((1124.328125, 1039.5135498046875), (110.39448547363281, 110.75920867919922), 149.56390380859375)

fitEllipseDirect():

((1124.2894287109375, 1039.5357666015625), (110.503173828125, 110.68196868896484), 144.33322143554688)

-------------------------------------

((1124.2652587890625, 1039.5408935546875), (110.40687561035156, 110.75054931640625), 146.06668090820312)

-------------------------------------

((1124.287841796875, 1039.5362548828125), (110.32542419433594, 110.75973510742188), 147.5096893310547)

-------------------------------------

((1124.2841796875, 1039.52197265625), (110.33760833740234, 110.75796508789062), 147.23269653320312)

-------------------------------------

((1124.252197265625, 1039.5379638671875), (110.44718170166016, 110.794189453125), 145.00491333007812)

**II Programmiercode**