Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Patrick-Alexander Mattich

Testroboter für Chip-Produkte mit Bahnsteuerung und optischer Kontrolle

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Patrick-Alexander Mattich

Testroboter für Chip-Produkte mit Bahnsteuerung und optischer Kontrolle

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter : Prof. Dr. Annabella Rauscher-Scheibe

Abgegeben am 14. September 2016

Patrick-Alexander Mattich

Thema der Bachelorthesis

Testroboter für Chip-Produkte mit Bahnsteuerung und optischer Kontrolle

Stichworte

Testhandler, OpenCV, Bilderkennung, Chiptest, Bestückungsmaschine, Roboter, Produktprüfung, Objekterkennung

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Entwicklung eines teilautomatisierten Testroboters für Halbleiterprodukte. Es wird eine Steuerungssoftware entwickelt, welche zwei Kameras zur Positionserkennung im Raum und für die Verifikation der zu testenden ICs benutzt. Die dafür notwendige Bilderkennung basiert auf der OpenCV Programmbibliothek. Das entwickelte Produkt soll für interne Produktprüfungen von der Firma TRINAMIC genutzt werden.

Patrick-Alexander Mattich

Title of the paper

Testhandler for semiconductor products with path control and computer vision

Keywords

test handler, OpenCV, image processing, chiptest, pick and place machine, robot, product tests, object recognition

Abstract

This thesis deals with the development of a partially automated test handler for semiconductor products. The developed control software uses two cameras for the position detection and for the part verification system. The object recognition software is based on the OpenCV library. The company TRINAMIC is going to use the finished handler for internal product tests.

Inhaltsverzeichnis

1. Einleitung

Automatische Testanlagen zur Kontrolle von ICs[1] sind heutzutage aus der Produktion von Halbleiterprodukten nicht mehr wegzudenken. Der Test jedes einzelnen produzierten ICs, ist bei den meisten Herstellern ein wesentlicher Bestandteil der Qualitätskontrolle. Nur große Halbleiterhersteller oder auf das Testen von ICs spezialisierte Firmen können sich solche teuren Testanlagen leisten. Kleinere Halbleiterentwickler lassen ihre Produkte häufig bei spezialisierten Anbietern testen. Dies rentiert sich jedoch nur bei größeren Stückzahlen. Für Vorserien und Prototypen erfolgen die Tests zumeist innerhalb des entwickelnden Unternehmens durch Handarbeit. Ziel dieser Bachelor Arbeit war es, einen preiswerten automatischen Testroboter für die Firma TRINAMIC zu entwickeln. Dieser würde eine einfach zu nutzende Alternative zum manuellen Testen sein und dadurch eine Arbeitserleichterung darstellen. Dabei stellt der LitePlacer [11], eine quelloffenen Bestückungsmaschine, den mechanischen Aufbau dar. Die Erkennung und Positionierung der ICs soll mit Hilfe einer Bilderkennungssoftware realisiert werden. Um eine einfache Bedienung zu ermöglichen, ist die Entwicklung einer graphischen Benutzeroberfläche vorgesehen.

1.1. Aufbau dieser Arbeit

Als Entwicklungsgrundlage dieser Arbeit wird in Kapitel 2 der aktuelle Stand der Firma TRINAMIC beschrieben sowie eine Anforderungsanalyse des Testhandlers durchgeführt.

Das Systemkonzept in Kapitel 3 stellt die zentralen Punkte dieser Arbeit vor und gibt einen Überblick über die notwendigen Entwicklungen.

Die zur Identifizierung der ICs benötigte Bilderkennung sowie die entwickelten Funktionen zur optischen Referenzierung im Raum werden in Kapitel 4 beschrieben.

In Kapitel 5 wird das Konzept und die Umsetzung der graphischen Benutzeroberfläche erläutert.

Der Aufbau des entwickelten Steuerprogramms für den Testhandler mitsamt dessen Eigenschaften werden in Kapitel 6 beschrieben.

[1] Integrierter Schaltkreis

Auf die Abläufe der Kalibrierung und die verwendeten Zustandsautomaten wird in Kapitel 7 eingegangen.

In Kapitel 8 werden verschiedene Funktionsnachweise durchgeführt und bewertet.

Um eine Übersicht der potentiellen Störungen zu geben, werden in Kapitel 9 verschiedene Arten von Fehlern erläutert und bewertet.

Abschließend wird in Kapitel 10 eine Zusammenfassung der Arbeit präsentiert und auf offen gebliebene Punkte eingegangen. Zusätzlich wird ein Ausblick auf mögliche Weiterentwicklungen der Hardware und Software gegeben.

2. Aktueller Stand

Das Unternehmen TRINAMIC ist eine sogenannte *Fabless Company*. Fabless, aus dem Englischen für *fabricationless*, bedeutet im Deutschen soviel wie *fabriklos*. Damit werden hauptsächlich Firmen aus der Halbleiterindustrie bezeichnet, welche ICs und andere Halbleiter entwickeln, jedoch keine eigenen Fertigungsstätten besitzen. Diese lassen ihre IC-Designs bei externen Dienstleistern fertigen. Dieses Geschäftsmodell bietet den Vorteil, dass sich kleinere Firmen die Entwicklung von ICs leisten können, ohne eine eigene Produktionsanlage aufbauen zu müssen. Diese kosten zum Teil mehrere Milliarden Euro [20]. Dadurch können sich die Unternehmen auf die Entwicklung der integrierten Schaltungen spezialisieren und die Einzelheiten der Fertigung auf andere Firmen auslagern. Ebenso gibt es Unternehmen, sogenannte Testhäuser, die das Testen von ICs anbieten. Bei Prototypen und Kleinserien wäre die Nutzung eines solchen Testhaus aufgrund der niedrigen Stückzahlen unwirtschaftlich. Diese werden bei TRINAMIC zurzeit manuell getestet und sortiert.

2.1. Bedarfsanalyse

Für das manuelle Testen muss ein Mitarbeiter folgende Arbeitsschritte für jeden einzelnen IC wiederholen:

- Den IC aus der Verpackung, dem sogenannten Tray, nehmen und in der richtigen Ausrichtung präzise in ein Testgerät legen.
- Den IC im Testgerät befestigen und den Testlauf starten.
- Auf die Auswertung des Tests warten.
- Die Testvorrichtung öffnen und den IC entsprechend der Auswertung in ein Tray einsortieren.

Je nach Gehäuseart des ICs variiert dementsprechend die Anzahl von ICs pro Tray. Das kleinste Gehäuse, welches TRINAMIC aktuell nutzt, ist ein *QFN32*[1] Gehäuse mit einer Abmessung von 5 x 5 mm. Die entsprechenden Trays[2] fassen dabei 576 ICs. Diese Anzahl von

[1] Quad Flat No Leads Gehäuse mit 32 Anschlüssen
[2] Vorratsbehälter für ICs

ICs manuell zu testen dauert viele Personenstunden. Der monotone Arbeitsablauf führt dabei zu Konzentrationsmangel und Unaufmerksamkeit. Dadurch können ICs falsch einsortiert oder verkehrt in die Testvorrichtung gelegt werden, was durch geringere Ausbeute zusätzliche Kosten verursacht. Eine Möglichkeit dem entgegenzuwirken, ist die Mitarbeiter immer nur für einen kurzen Zeitraum ICs testen zu lassen, was jedoch den gesamten Testablauf verlängert. Dadurch ist diese Art zu testen langsam und zum Teil fehleranfällig. Industrielle Testanlagen sind hingegen wesentlich schneller. Diese können, je nach Ausführung, mehr als 10000 ICs pro Stunde testen [13].
Die Verwendung eines automatisierten Testroboters würde eine Entlastung der Mitarbeiter ermöglichen. Es könnten alle ICs ohne Pause getestet werden und Fehler durch Konzentrationsmangel wären ausgeschlossen. Kommerziell erhältliche Testanlagen sind jedoch zu teuer, um eine Anschaffung für die kleinen Stückzahlen der Prototypen und Kleinserien zu rechtfertigen. Deshalb soll ein preiswerter und automatischer Testhandler auf Basis von TRINAMIC Komponenten entwickelt werden, der die anfallenden Mengen von bearbeiten kann. Der Durchsatz an ICs steht dabei nicht im Vordergrund, sondern die Arbeitserleichterung der Mitarbeiter.
Zusätzlich soll der Testhandler als Demonstrationsobjekt für die Anwendung von TRINAMIC Produkten dienen, zum Beispiel auf Messen.

2.2. Anforderungsanalyse

Die durch TRINAMIC gestellten Anforderungen an die Entwicklung des Testhandlers lassen sich in funktionale, nichtfunktionale und technische Anforderungen einteilen. Diese sind im Folgenden wiedergegeben.

2.2.1. Funktionale Anforderungen

Funktionale Anforderungen charakterisieren gewünschte Eigenschaften eines zu entwickelnden Produktes. Sie sind spezifisch für das Produkt und beschreiben was es machen soll.

- Die zu testenden ICs müssen aus einem Vorratstray genommen und nach dem Testen je nach Testergebnis in Trays sortiert werden.
- Die ICs müssen mit ausreichender Präzision in die Trays und den Testsockel gelegt werden.
- Die Software muss eine externe Testeinheit mit Testsockel ansteuern und auswerten können.
- Die Trays müssen lückenlos aufgefüllt werden.

- Die ICs müssen in der richtigen Ausrichtung in die Trays gelegt werden.
- Der Testablauf muss jederzeit abgebrochen werden können.
- Es muss die Möglichkeit geben, Hardware spezifische Parameter einstellen zu können.
- Es müssen Konfigurationen zum testen von verschiedenen IC-Typen erstellt werden können.
- Die Software muss eine graphische Benutzeroberfläche haben.
- Die Erkennung der Ausrichtung und Position des zu testenden ICs muss über einen optischen Kontrollmechanismus erfolgen.
- Die Referenzierung der Position muss über optische Markierungen erfolgen.

2.2.2. Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen sind Eigenschaften, welche das zu entwickelnden Produkt haben soll. Dabei beziehen diese sich darauf, wie die gewünschten Eigenschaften umgesetzt werden sollen oder Bedingungen, unter denen diese ausgeführt werden sollen.

- Die Benutzeroberfläche soll leicht verständlich und bedienbar sein.
- Die Benutzeroberfläche soll aktuelle Positionswerte und Testergebnisse darstellen.
- Die Software soll von TRINAMIC wartbar sein und bei Bedarf erweitert werden können.
- Die Wahl der Steuerelektronik soll unabhängig von der Software sein. Die Elektronik soll modular wählbar sein.
- Hardware und IC-Modell spezifische Parameter sollen über Konfigurationsdateien geladen werden können.
- Es soll der vorgegebene Programmierstil von TRINAMIC genutzt werden.

2.2.3. Anforderungen an die technische Realisierung

Die Anforderungen an die technische Realisierung geben vor, welche Arten von technischen Ressource verwendet werden sollen und wie diese verarbeitet werden sollen.

- Es soll die quelloffene Bestückungsmaschine LitePlacer als mechanische Grundlage verwendet werden.

- Die Software muss in C++ entwickelt werden, um die Kompatibilität mit bestehender TRINAMIC Software zu gewähren.
- Die Software muss als Plug-in[3] für die TMCL-IDE (siehe Abschnitt A.2.1) entwickelt werden.
- Für die Entwicklung der Benutzeroberfläche muss das Qt-Framework genutzt werden.
- Der Testhandler muss für die Ansteuerung der Motoren TRINAMIC Module verwenden.
- Die Software muss unter auf einem *Microsoft Windows* Betriebssystem lauffähig sein.

[3] optionales Software-Modul

3. Systemkonzept

Ziel der Entwicklung des Testhandlers soll eine Zeit und Kosten Ersparnis gegenüber dem manuellen Testen, Programmieren und Sortieren sein. Da aus Kostengründen die Verwendung von speziell angefertigten Maschinen nicht möglich ist, wird eine günstige Bestückungsmaschine die mechanische Grundlage sein. Diese bieten einen ähnlichen Aufbau wie manche kommerziell erhältlichen Testroboter. Falls notwendig, kann der Bestückungsautomat an die benötigten Funktionen angepasst werden. Um eine Positionierung in den eng tolerierten Trays und Testsockeln zu ermöglichen, soll eine optische Kontrollmethode auf Basis von Kameras verwendet werden, wozu die Verwendung von Bildverarbeitungssoftware notwendig ist. Für die elektronischen Komponenten werden vorwiegend TRINAMIC Produkten verwendet. Die Steuerung soll in die vorhandene TRINAMIC TMCL-IDE als Plug-in integriert werden, um eine möglichst einfache Wartung und Weiterentwicklung seitens TRINAMIC zu ermöglichen. Da ein multifunktionaler Einsatz des Testhandlers vorgesehen ist, sollen für verschiedene IC-Typen Testparameter angelegt werden können. Am Ende der Entwicklung sollen komplette Trays eines IC Typen automatisch getestet und sortiert werden können, wobei eine korrekte Orientierung der ICs im Testsockel sowie den Trays erforderlich ist.

3.1. Verwendete Hardware

Der mechanische Aufbau ist ein in drei Dimensionen verfahrender Flächenportalroboter, dessen Effektor[1] mittels einer vierten Achse um die Z-Achse rotiert werden kann. Er basiert auf der quelloffenen Bestückungsmaschine LitePlacer. Für den Einsatz in diesem Projekt wurde die X-Achse im verfahrbaren Weg verdoppelt (siehe Abschnitt B.1.1). Als Effektor wird ein steuerbarer Saugheber[2] verwendet. Die Arbeitsfläche ist in mehrere Bereiche aufgeteilt:

- Testeinheit:
  Für die Aufnahme der eigentlichen IC-Testelektronik befindet sich eine Aussparung

[1] Ausgerüstetes Werkzeug
[2] Vakuumpipette

mittig in der Arbeitsfläche. Um den Testsockel auf gleicher Höhe positionieren zu können wie die Arbeitsfläche, wird die Testelektronik versenkt eingebaut, siehe Punkt 1 in Abb. F 3.1.

- Kameras:
  Für die Erkennung des zu testenden ICs ist eine aufschauende Kamera unterhalb der Arbeitsfläche montiert (siehe Abschnitt B.3), die sogenannte Tischkamera (siehe Punkt 2 in Abb. 3.1). Eine zweite Kamera ist neben dem Saugheber des Testhandler befestigt und schaut auf die Arbeitsfläche. Sie wird mit mit dem Saugheber mitbewegt (siehe Punkt 6 in Abb. 3.1) und erkennt Referenzmarker.

- Trayhalterung:
  Um die Trays reproduzierbar und präzise anfahren zu können muss ein Verrutschen dieser verhindert werden. Dafür wurden Schablonen in die Arbeitsfläche integriert in denen die Trays befestigt werden können (siehe Punkt 3 in Abb. 3.1 und Abschnitt B.2.1). Zusätzlich befinden sich auf den Schablonen optische Marker für die Kameraerkennung zur Referenzierung der Trays (siehe Abschnitt 3.3).

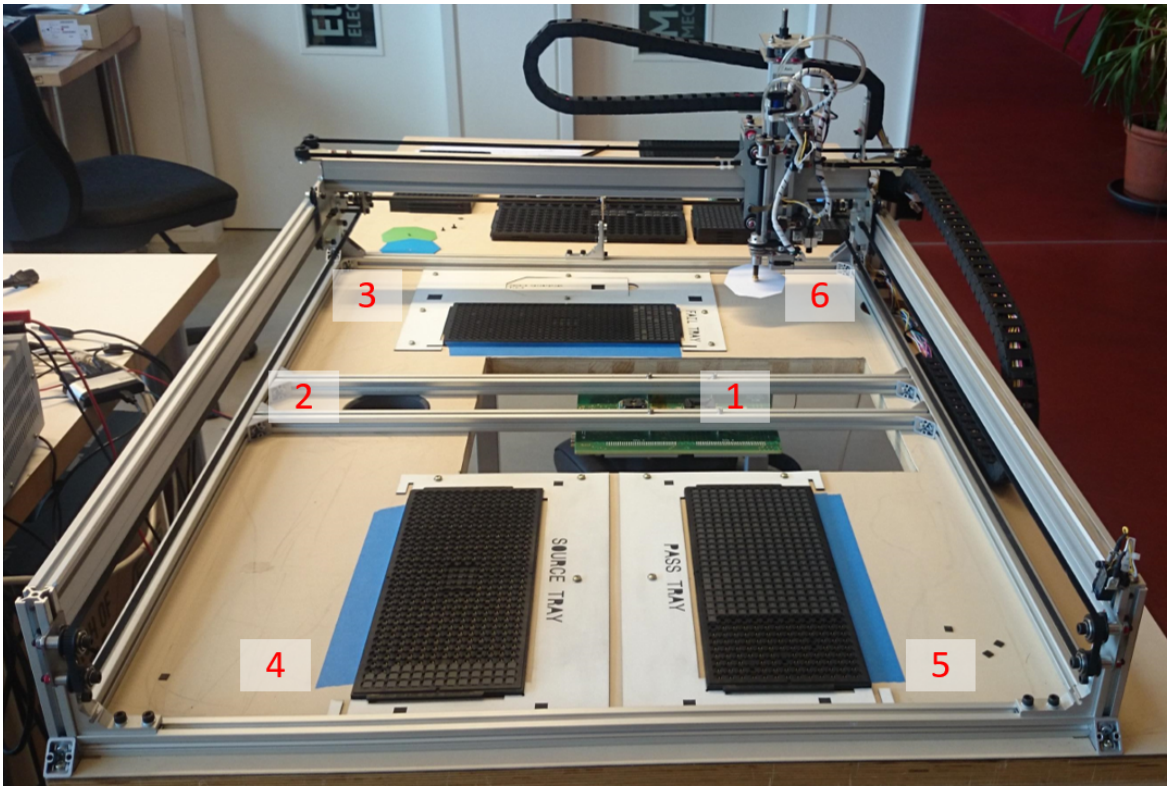


Abbildung 3.1.: Mechanischer Aufbau des Testhandlers. (1) ist der Testsockel, (2) ist die von unten schauende Tischkamera, (3) ist das Failtray, (4) ist das Sourcetray, (5) ist das Passtray, (6) ist der Saugheber

Für die Ansteuerung der vier Achsen, sowie des Saughebers wird das TMCM-6210 Modul, eine industrielle Steuerelektronik von TRINAMIC genutzt (siehe Abschnitt A.2.4). Für die Bilderkennung werden zwei Kameras und eine einstellbare Lichtquelle zur Ausleuchtung verwendet. Die Testelektronik inklusive Testsockel wird von TRINAMIC gestellt (siehe Abschnitt A.3).
Der elektronische Aufbau ist in Abb.3.2 als schematischer Schaltplan dargestellt. Weitere Ausführungen über den elektronischen und mechanischen Aufbau sind in Abschnitt A und B nachzulesen.

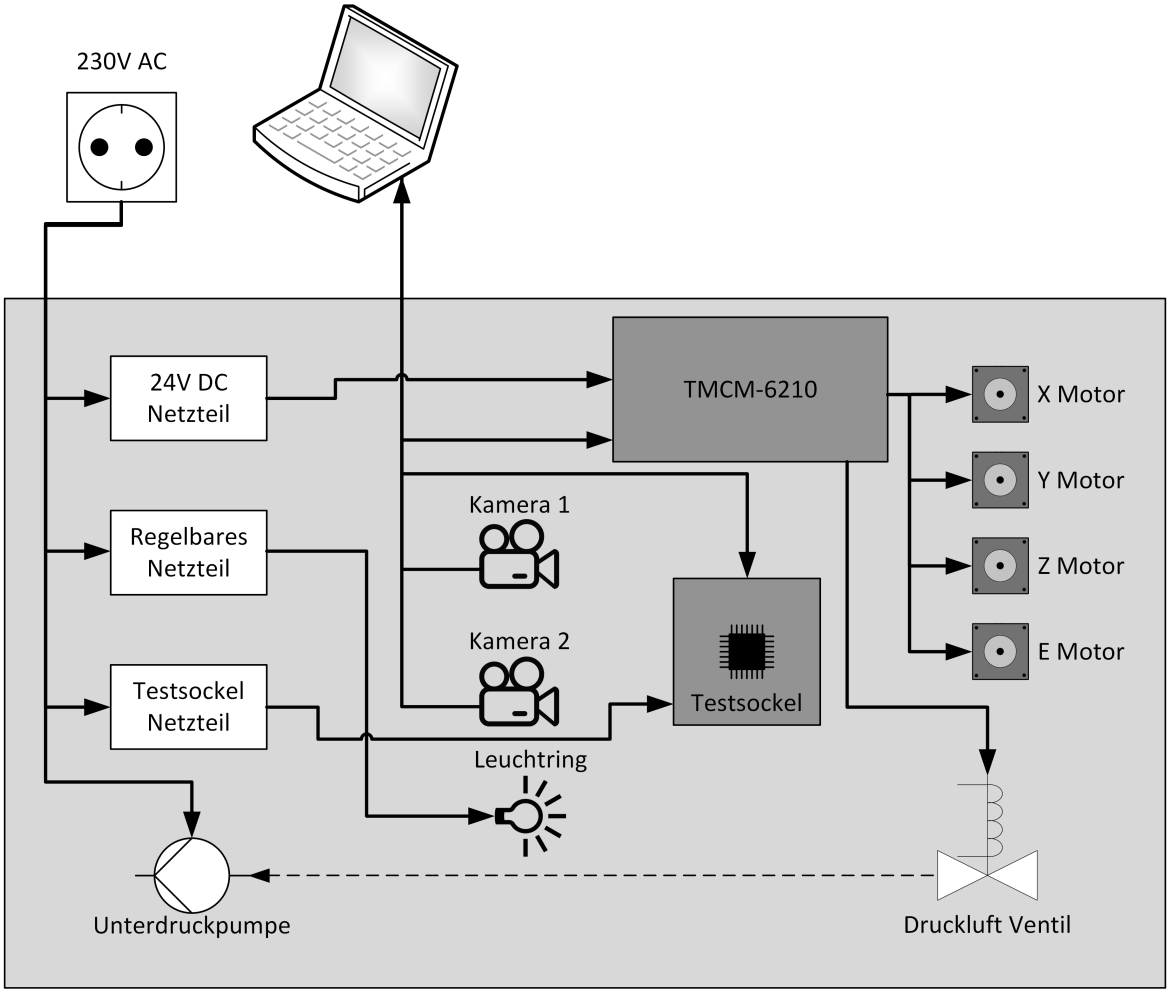


Abbildung 3.2.: Schematische Verschaltung des Testhandlers

3.2. Vergleich des LitePlacers mit kommerziellen Testanlagen

Im Vergleich zu erhältlichen industriellen Testanlagen ergeben sich folgende Unterschiede:

- Die kommerziellen Testhandler sind deutlich präziser und stabiler gebaut, als der LitePlacer, womit sich eine höhere Wiederholgenauigkeit und Winkeltreue ergibt.
- Durch die stabilere Konstruktion sind auch höhere Bewegungsgeschwindigkeiten möglich, ohne dass es zu Einbußen der Präzision kommt. Das hat einen höheren Durchsatz von ICs zur Folge.
- Durch eine genau auf den Test von ICs angepasste Entwicklung, können zusätzliche Verbesserungen wie einen automatischen Wechsel der Trays oder eine direkte Markierung der defekten ICs eingebaut werden.
- Der LitePlacer inklusive den Modifikationen ist wesentlich günstiger[11].

3.3. Zwei-Kamera-Prinzip

Das Grundprinzip der genutzten Bilderkennung bildet die Verwendung für zwei Kameras, die jeweils unterschiedliche Funktionen erfüllen. Grundsätzlich werden für den Testhandler zwei Arten der Positionserkennung benötigt, welche im Folgenden beschrieben werden.

3.3.1. Die Erkennung der Lage im Raum

Um Zielkoordinaten genau berechnen und anfahren zu können muss die aktuelle Position bekannt sein. Dafür bieten sich zwei Möglichkeiten an, ein Closed-Loop oder ein Open-Loop System. Bei einem Closed-Loop System erfolgt ein Messung der realen Position, wodurch Positionsfehler ausgeglichen werden können. Bei einem Open-Loop System hingegen erfolgt keine Auswertung der realen Position. Sollte es hier zu einem Positionsfehler kommen, zum Beispiel durch die Blockade des Motors, wird dies nicht erkannt. Durch den entstandenen Versatz ist keine genaue Positionierung mehr möglich, da die theoretischen und realen Koordinaten nicht mehr übereinstimmen.
Durch die Vorgabe von TRINAMIC ein Open-Loop System zu verwenden bedarf es einem Verfahren, mit dem dennoch eine ausreichend hohe Präzision zu erreichen ist. Klassischerweise werden in Open-Loop System Endschalter genutzt, die aktiviert werden, wenn der

Anfang oder das Ende der verfahrbaren strecke erreicht wird. Durch Aktivierung eines Endschalters wird die aktuelle Position erkannt und es können Bahnberechnung erfolgen. Dieser Vorgang ist das sogenannte Homing (siehe auch Abschnitt 7.1.1).

Um von der Homingposition, meistens dem Nullpunkt des Systems, zu einem mechanisch festgelegten Ort zu fahren, muss der genaue Abstand von dem Nullpunkt zu dem Ziel bekannt sein. Theoretisch wäre ein Ausmessen des Abstandes mithilfe von Messinstrumenten möglich, praktisch ist das aufgrund des mechanischen Aufbaus solcher Anlagen (kein Anschlag, ungenaue Abstände zwischen Endschaltern) meistens nur sehr ungenau durchführbar. Der hier gewählte Ansatz dieses Problem zu umgehen, basiert auf einer optische Erkennung von Markierungen, anhand derer sich der Testhandler referenziert. Zur Erfassung der Markierungen wird eine, auf die Arbeitsfläche gerichtete, Kamera neben dem Saugheber mitgeführt. Es reicht einen Marker näherungsweise anzufahren. Sobald dieser im Blickfeld der Kamera ist, findet eine Ausrichtung anhand der Marker statt. Dadurch muss nur die Differenz zwischen Marker und Zielposition vermessen worden sein. In die Tray-Halterungen sind derartige Marker integriert (siehe Punkt 3 in Abb. 3.1). Da der Abstand von dem Marker zu dem Tray bekannt ist, kann ein automatisches Einmessen der Traypositionen erfolgen. Da Konstruktionsbedingt eine räumliche Differenz zwischen dem Saugheber und der Kamera liegt (siehe Abb. 3.3), muss dieser Versatz bekannt sein und ausgeglichen werden, wenn ein mit der Kamera anvisiertes Objekt angehoben werden soll.

3.3.2. Die Erkennung der Ausrichtung und Position des ICs

Um einen IC in einen Testsockel oder Tray einzulegen, muss er zum einen die richtige Ausrichtung (Rotation der Z-Achse) haben und zum anderen muss er präzise genug eingesetzt werden um ein verkanten und damit eine Beschädigung des Testsockels und des ICs zu vermeiden. Um den Fehler von falsch eingelegten oder fehlenden ICs auszuschließen erfolgt zusätzlich eine Kontrolle auf Versatz in X- und Y-Richtung und auf die Ausrichtung. Die Erkennung der Ausrichtung kann je nach Gehäuseart des ICs von der Gehäuseober- oder Unterseite ermittelt werden. Auf der Oberseite ist in der Regel eine Ecke des ICs mit einem kleinen Punkt markiert. Auf der Unterseite ist die Ausrichtung entweder durch die Anordnung der Anschlüsse oder einer Markierung, wie zum Beispiel eine abgeschrägte Ecke, erkennbar. Durch Erkennung der Außenkanten des ICs lässt sich zudem der Mittelpunkt und damit den Versatz in X- und Y-Richtung bestimmen. Grundsätzlich wäre eine Erkennung der Ausrichtung durch eine Bildanalyse der Oberseite eines ICs möglich. Jedoch liegen die ICs mit einem zumeist matt schwarzen Gehäuse in einem matt schwarzen Tray, wodurch sich ein schwacher Kontrast ergibt, was eine genau Erkennung eines ICs in einem vollständig bestückten Tray erschwert. Deshalb wurde die Methode gewählt, jeden zu testenden IC einzeln über eine Kamera zu halten und anhand der Unterseite die Position und Ausrichtung zu bestimmen (siehe Abschnitt 4.6). Das hat den Vorteil, dass über dem Saugheber ein

heller Hintergrund angebracht werden kann, um den Kontrast zwischen IC und Umgebung zu verstärken. Auch eine Überprüfung, ob der IC tatsächlich angehoben wurde, ist dadurch möglich (siehe Abschnitt 4.6.3).
Für beide Verfahren wird jeweils eine eigene Kamera benötigt, wobei die eine stationär verbaut und die andere neben dem Saugheber mitgeführt wird. Das bietet den zusätzlichen Vorteil der Anpassung der Kamera an die entsprechende Funktion. So muss die mitzuführende Kamera kompakt sein, damit der Saugheber nicht behindert wird (siehe Abschnitt 4.2). Die Kamera zur Erkennung der Unterseite muss eine optische Vergrößerung beinhalten, um die einzelnen Anschlüsse der ICs noch erfassen zu können. Mit Hilfe einer anpassbaren Beleuchtung können Reflexionen der metallenen Anschlüsse vermieden werden (siehe Abschnitt 4.8).

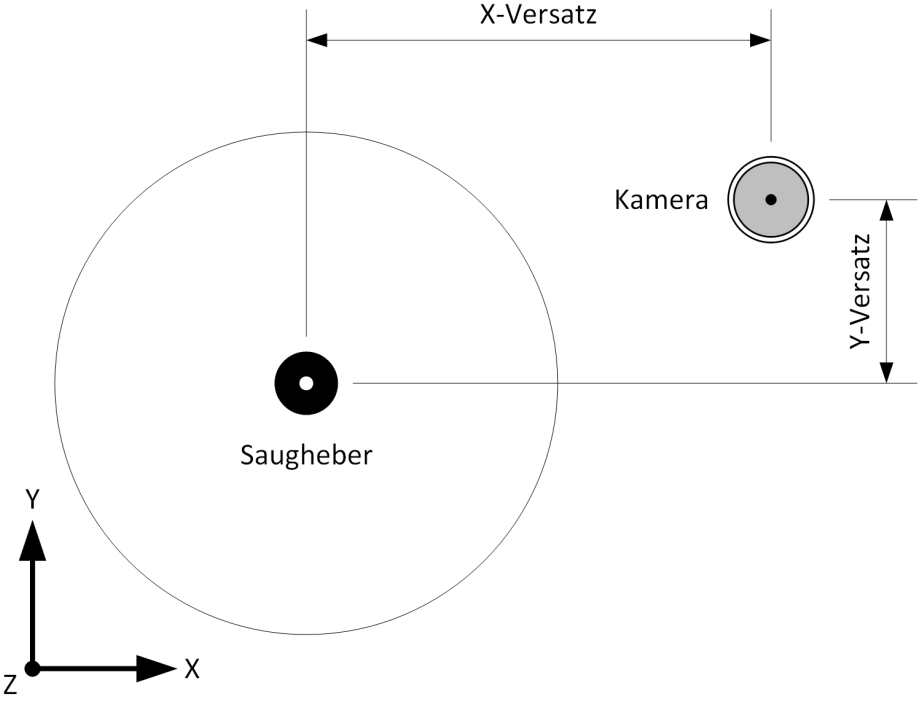


Abbildung 3.3.: Zu sehen ist der Versatz zwischen dem Saugheber und der Kamera in X- und Y-Richtung (Blick von unten)

3.4. Interaktion der Software

Für die Software des Testhandlers werden folgende Elemente benötigt:

- Ansteuerung der Elektronik

- Bilderkennung und Verarbeitung
- Benutzeroberfläche
- Ablaufsteuerung

Da die Integration der Software des Testhandlers als Plug-in in die TMCL-IDE von TRINAMIC vorgegeben wurde, erfolgt die Ansteuerung der Elektronik über deren vorgefertigten Schnittstellen (siehe Abschnitt A.2.1). Zusätzlich wird die gleiche Entwicklungsumgebung und Programmiersprache genutzt, um eine möglichst große Kompatibilität und einfache Wartung zu ermöglichen. Aus diesem Grund werden die Sprache C++ und die Entwicklungsumgebung Qt Creator verwendet. Diese ermöglicht eine Entwicklung von Benutzeroberflächen (siehe Abschnitt 5.1). Für die Bildverarbeitung wird OpenCV verwendet, eine quelloffene Programmbibliothek , welche Funktionen und Algorithmen für die Bildverarbeitung und Erkennung enthält und zusätzlich die Einbindung und Auswertung von Kameras ermöglicht (siehe Abschnitt 4.1). Aufgrund der Verfügbarkeit als Bibliothek kann die Bilderkennung in ein Softwareprojekt integriert werden und muss nicht als ein externes Programm aufgerufen werden. In Abb. 3.4 ist ein Überblick über den Zusammenhang der einzelnen Softwareelemente dargestellt.

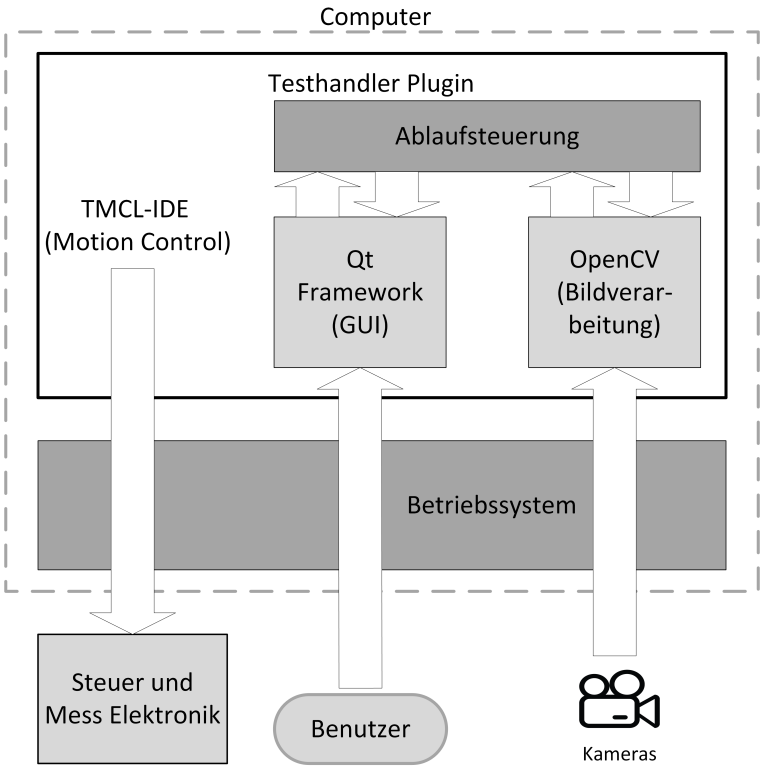


Abbildung 3.4.: Überblick der verwendeten Software

4. Digitale Bildverarbeitung

Die digitale Bildverarbeitung ist eine Disziplin der Informatik und Elektrotechnik. Sie befasst sich mit der Verarbeitung von Bildern in Form von digitalen Signalen, womit ist sie ein Teil der digitalen Signalverarbeitung ist. Häufige Ziele der Bildverarbeitung sind die Gewinnung bestimmter Informationen oder die Generierung und Modifikation von Bildern. Dabei gibt es in der Theorie keinen Unterschied, ob einzelne Photos oder Bilder aus einem Video verwendet werden. Da die Bilder aus diskreten Elementen (Pixel) bestehen, können Methoden der digitalen Signalverarbeitung angewandt werden. Dabei wird das Bild als zweidimensionales Signal behandelt. Ein Teilbereich der Bildverarbeitung ist die Bilderkennung, welche sich mit der Erkennung bestimmter Muster oder Objekte in Bildern befasst.
Häufige Anwendungsfälle der digitalen Bildverarbeitung sind unter anderem:

- Medizintechnik
- Optische Qualitätskontrolle
- Optische Messverfahren
- Robotik

4.1. Bildverarbeitungsbibliothek OpenCV

OpenCV ist eine quelloffene Programmbibliothek für C, C++, Python und andere Programmiersprachen. Sie enthält Funktionen und Algorithmen zur Bildverarbeitung und Bilderkennung sowie Bibliotheken für Maschinelles Lernen. OpenCV wurde im Januar 1999 veröffentlicht und seitdem stetig weiterentwickelt. Ursprünglich von der Firma *Intel* entwickelt, wird OpenCV heutzutage von der Firma *Itseez* weiterentwickelt. In dieser Arbeit wurde die OpenCV Version 2.4.11 verwendet. Da in diesem Kapitel nur auf die in dieser Arbeit verwendeten Funktionen von OpenCV eingegangen wird, sei für einen tieferen Einblick auf die Dokumentation in [3] verwiesen.

4.1.1. Datentyp `Mat`

Um in OpenCV ein Bild Laden, Speichern und Bearbeiten zu können, muss es als verwendbares Dateiformat vorliegen. Dafür beinhaltet OpenCV den Datentyp `Mat`. Dieser besteht aus zwei Teilen:

1. Aus einer oder mehreren zweidimensionalen Matrizen. Die Elemente der Matrizen entsprechen jeweils einem Pixel des gespeicherten Bildes. Die Anzahl der Matrizen ergibt sich aus der Menge der Farbkanäle eines Bildes. So hat zum Beispiel ein Graustufenbild nur eine Matrix und ein Farbbild drei Matrizen, jeweils eine für Rot, Grün und Blau. Zusätzlich ist der Datentyp der Pixel auswählbar. Der kleinste ist ein *char* mit 8 Bit, was einer Auflösung von 256 Abstufungen pro Farbe entspricht. Bei einem RGB-Bild[1] ergeben sich damit insgesamt $(2^8)^3$ = 16.777.216 mögliche Farben. In Abb. 4.1 und in Abb. 4.2 ist ein Beispiel für ein einkanaliges Graustufenbild mit 5 x 5 Pixeln und 8 Bit Auflösung dargestellt.
2. Aus einem Header, der Informationen wie die Größe der Matrix, die Speicheradresse und andere Parameter enthält.

Der Speicher wird automatisch und dynamisch allokiert und deallokiert. Dadurch lassen sich zur Laufzeit in einem `Mat`-Objekt verschieden große Bilder Speichern, ohne vorher eine Größe festzulegen. Die Umwandlung von einem Grafikformat, wie beispielsweise von JPEG[2], in Matrizen erfolgt automatisch beim Laden des Bildes per `imread(Dateipfad)` Funktion (siehe Listing 4.1). Die Elemente der Matrizen lassen sich einzeln oder anhand von Funktionen wie Filtern in der Gesamtheit des Bildes manipulieren.

Listing 4.1: Beispiel für *imread()* Funktion

```
1 Mat image = imread("C:\\Bilder\\testbild.jpeg");
```

.

[1] Farbbild, welches die dargestellten Farben aus Rot, Grün und Blau erzeugt

[2] Joint Photographic Experts Group, ein verbreitetes Bildformat

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 128 & 255 & 255 & 255 & 255 \\ 128 & 128 & 255 & 255 & 255 \\ 0 & 128 & 128 & 255 & 255 \\ 0 & 0 & 128 & 128 & 255 \\ 0 & 0 & 0 & 128 & 128 \end{pmatrix} \end{array}$$

Abbildung 4.1.: Matrix für ein Graustufenbild mit 5 x 5 Pixel bestehend aus dem Datentyp *char*

.

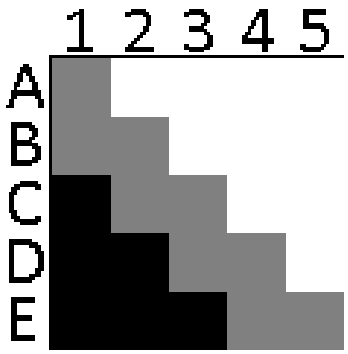


Abbildung 4.2.: Darstellung der Matrix aus Abb. 4.1 als Bild

.

4.1.2. Integration von Kameras

OpenCV ermöglicht es neben Bilder auch Videos zu öffnen und einzubinden. Dabei spielt es keine Rolle, ob es sich bei der Videoquelle um eine Datei oder um eine angeschlossene Kamera handelt. Nur die Quellenangabe unterscheidet sich zwischen einem Dateipfad für eine Datei, oder einer ID[3] von einer angeschlossenen Kamera. Um Bilder aus einem Video bearbeiten zu können, muss ein Einzelbild aus dem Video in eine `Mat` Objekt gespeichert werden. Diese kann dann von den Funktionen und Algorithmen von OpenCV verwendet werden. Ein Beispielprogramm ist in Listing 4.2 dargestellt. In diesem wird die Kamera mit der *ID = 0* geöffnet und und das Bild in die Variable `frame` gespeichert. Durch die Funktion

[3] Identifikator

`imshow(title, image)` wird ein Fenster mit dem Titel "Bild von Kamera"geöffnet und der Inhalt von `frame` angezeigt.

Die Einbindung mehrere Kameras ist durch die Angabe mehrere IDs und Erzeugung mehrerer `VideoCapture` Elemente möglich. Die IDs sind nicht von der Hardware der Kamera abhängig, sondern hängen unter anderem davon ab, an welchen Anschluss und in welcher Reihenfolge die Kameras an einen Computer angeschlossen wurden. Wenn ein Video verwendet werden soll, muss periodisch ein neues Bild eingelesen werden.

Listing 4.2: Öffnen einer Kamera und Darstellen des Bildes

```
1  VideoCapture source;
2  Mat frame;
3
4  source  = VideoCapture( 0 );
5  if(!source.isOpened())  // Pruefen, ob eine Kamera geoeffnet werden konnte
6  {
7      cout << "Kann Kamera ( 0 ) nicht oeffnen!" ;
8  }
9  else
10 {
11     source >> frame; // Ein neues Bild von der Kamera in "frame" laden
12     imshow("Bild von Kamera", frame);
13 }
```

4.1.3. Verwendete Bibliotheksfunktionen

Die entwickelte Software des Testhandlers nutzt für die Bilderkennung und -verarbeitung verschiedene Funktionen der OpenCV Bibliotheken, welche im Folgenden kurz erläutert werden:

- `RotatedRect` Klasse
  Diese Klasse repräsentiert ein rotiertes, also nicht waagerechtes, Rechteck. Es ist über seinen Mittelpunkt, seine Höhe und Breite sowie über seinen Rotationswinkel definiert.
- `Rect` Klasse
  Diese Klasse repräsentiert ein Rechteck, welches aus einer Ursprungskoordinate und der Höhe und Breite besteht. Eine Rect kann nicht rotiert sein.
- `cvtColor`
  Wandelt den Farbraum eines Bildes in einen anderen um, zum Beispiel ein Farbbild in ein Graustufenbild.
- `threshold`
  Diese Funktion binarisiert eine einkanaliges Bild. Das heißt, es wandelt beispielsweise ein Graustufenbild in ein Schwarz-Weiß-Bild um. Der Schwellwert muss angegeben

werden. Es kann zwischen mehreren Schwellenwertverfahren [5, S.135-138] gewählt werden. Zusätzlich kann eine adaptiven Schwellwertbestimmung nach Otsu [22] gewählt werden, die einen Schwellwert abhängig vom Bild berechnet.

- `findContours`
  Erkennt in einem einkanaligen Bild Konturen von Objekten. Genutzt wird dafür ein Algorithmus von Suzuki und Abe [23]. Jede Kontur eines Objekts wird als Array von Koordinaten gespeichert. Es kann zwischen mehreren Erkennungsmodi gewählt werden.

- `contourArea`
  Berechnet die Anzahl der Pixel, die sich innerhalb einer angegebenen, geschlossenen Kontur befinden.

- `minAreaRect`
  Findet das kleinstmögliche Rechteck, welches eine angegeben Menge von Koordinaten, zum Beispiel eine Kontur, umspannt.

- `resize`
  Diese Funktion kann die Dimension (Höhe und Breite) des Bildes ändern. Dabei kann zwischen verschiedenen Interpolations Algorithmen gewählt werden.

- `getRotationMatrix2D`
  Erzeugt mithilfe eines angegebene Winkels und Drehpunkts eine Rotationsmatrix mit der Bilder rotiert werden können.

- `warpAffine`
  Ermöglicht eine Transformation eines Bildes mithilfe einer Transformationsmatrix, zum Beispiel einer Rotationsmatrix.

- `flip`
  Ermöglicht ein Spiegeln des Bildes in horizontaler und vertikaler Richtung.

- `inRange`
  Überprüft, ob in einem Bild Farben existieren, welche zwischen einem unteren und einem oberen Schwellwert liegen. Alle Pixel die innerhalb der Schwellwerte liegen werden weiß, alle anderen schwarz eingefärbt.

- Allgemeine Zeichenfunktionen:
  Dazu zählen Funktionen wie `line` zum Einzeichnen von Linien, `circle` zum Einzeichnen eines Kreises oder `rectangle` zum Zeichnen eines Rechteckes.

4.2. Auswahlkriterien der Kameras

Die Eigenschaften der Kameras müssen den unterschiedlichen Anforderungen entsprechen, die durch den gegebenen Einsatzbereich vorgegeben sind. Im folgenden werden die Auswahlkriterien der Fahr- und Tischkamera erläutert.

4.2.1. Fahrtkamera

Die Fahrtkamera wird genutzt, um die optischen Marker auf der Arbeitsfläche zu erkennen. Da sie mit dem Saugheber mitbewegt wird, ist eine Anforderung, dass sie leicht und klein gebaut ist, um die Bewegungen des Testhandlers nicht zu beeinträchtigen. Eine besonders hohe Auflösung der Kamera ist nicht notwendig, da keine feinen Details erkannt werden müssen, sonder die Außenkante eines mindestens 1 x 1 cm großen Quadrates. Wichtig ist hingegen, dass keine automatische Fokusfunktion vorhanden ist. Diese würde zwei Nachteile mit sich bringen. Zum einen kostet das Fokussieren Zeit, zum anderen wäre dadurch eine mögliche Fehlerquelle gegeben, wenn sich der Autofokus falsch einstellen sollte. Durch den mechanischen Aufbau des LitePlacers ist ein konstanter Abstand der Kamera zu der Arbeitsfläche gegeben. Dadurch ist ein manuell eingemessener, fest eingestellter Fokuspunkt möglich. Um unabhängig von dem Umgebungslicht die Markierungen erkennen zu können, ist eine Beleuchtung nötig, die mit der Kamera mitgeführt wird. Die hier verwendete Kamera ist eine preisgünstige USB-Endoskop-Kamera mit einer Auflösung von 640 x 480 Pixeln, einem festen Fokuspunkt und einer integrierten Beleuchtung. Das Gehäuse ist rund, mit einem Durchmesser von 7 mm und 45 mm Länge. Aufgrund dieser geringen Abmessungen ist ein einfacher Einbau möglich. Wegen des mechanischen Aufbaus des LitePlacers besteht ein Versatz in X- und Y-Richtung zwischen der Fahrtkamera und dem Saugheber (siehe Abb. 4.3). Dieser muss ausgeglichen werden, um ihn auf die Position der Kamera zu fahren. Der Saugheber befindet sich deshalb nicht im Sichtfeld der Fahrtkamera. Dadurch kann mittels der Kamera keine direkte Kontrolle erfolgen, ob ein IC wirklich angehoben wurde.

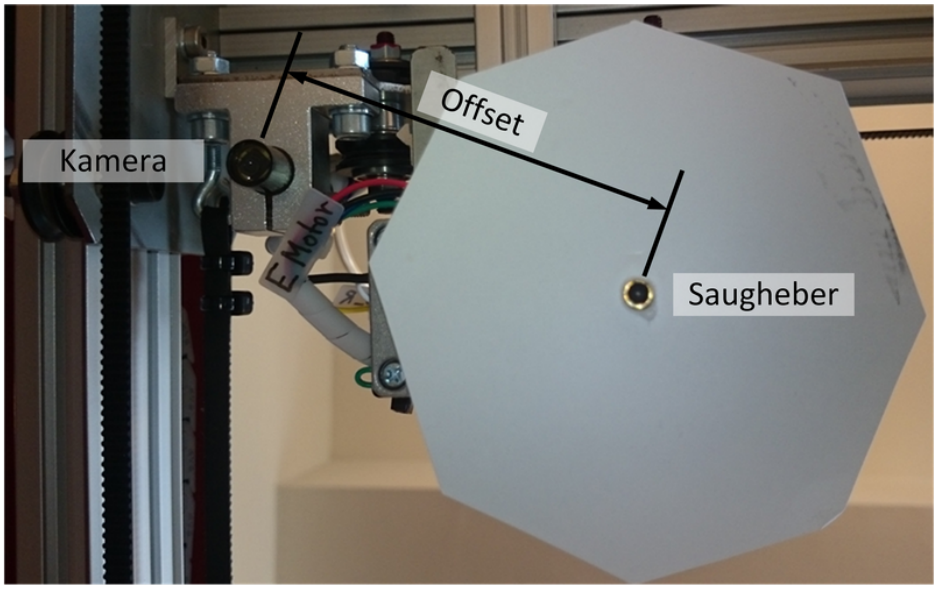


Abbildung 4.3.: Sicht von unten: Räumlicher Versatz in X- und Y-Richtung zwischen Fahrtkamera und Saugheber.

4.2.2. Tischkamera

Diese Kamera soll die Unterseite des ICs filmen, um die Ausrichtung und den Versatz der ICs festzustellen. Da die Orientierung des ICs anhand markanter Stellen der Anschlussflächen erfolgt, muss die Kamera diese Stellen genau genug abbilden können. Die kleinsten Strukturen, die gut erkennbar sein müssen, sind durch die ICs von TRINAMIC vorgegeben. Die ICs mit dem kleinstmöglichen Pinabstand und den geringsten Außenkantenlänge haben aktuell die QFN48 7 x 7 mm und QFN32 5 x 5 mm Gehäuse. Bei beiden sind die Anschlussflächen 0.25 mm breit und der Abstand zwischen den einzelnen Anschlüssen beträgt ebenfalls 0.25 mm. Um in dieser Größenordnung ein verwertbares Bild zu erhalten, ist die Verwendung einer optischen Vergrößerung notwendig. Weiterhin ist aus den selben Gründen wie bei der Fahrtkamera ein fest einstellbarer Fokuspunkt von Vorteil. Im Gegensatz zu der Fahrtkamera ist das Gewicht und die Größe nicht relevant, da die Tischkamera stationär verbaut wird. Gewählt wurde eine USB-Mikroskopkamera [8], welche eine Auflösung von 2592 x 1944 Pixel und einen fest einstellbaren Fokus hat. Um eine schnellere Berechnung zu ermöglichen wurden allerdings nur Bilder mit einer Auflösung von 1600 x 1200 Pixel verwendet, wodurch die Anzahl der Pixel pro Bild von 5.038.848 auf 1.920.000 sinkt. Dadurch verringert sich dementsprechend die Anzahl der Pixel pro Millimeter, wodurch sich die Erkennung von Details verschlechtert. So wird bei einem IC-Gehäuse mit einer Pinbreite von 0.25 mm (zum Beispiel ein QFN48 7 x 7 Gehäuse), die Anzahl der Pixel je Pin von 14,46 Pixel auf 8,92 reduziert. Dies reicht zur Erkennung der Orientierung des ICs jedoch aus. Die

optisch Vergrößerung ist stufenlos einstellbar. Die in die Kamera eingebaute Beleuchtung ist jedoch nicht nutzbar, da sie nur ein sehr direktes Licht produziert (siehe Kapitel 4.8). Für die Tischkamera wurde eine Halterung entworfen und 3D gedruckt, die Details dazu finden sich im Anhang unter B.3.

4.3. Kantenerkennung von rechteckigen Objekten

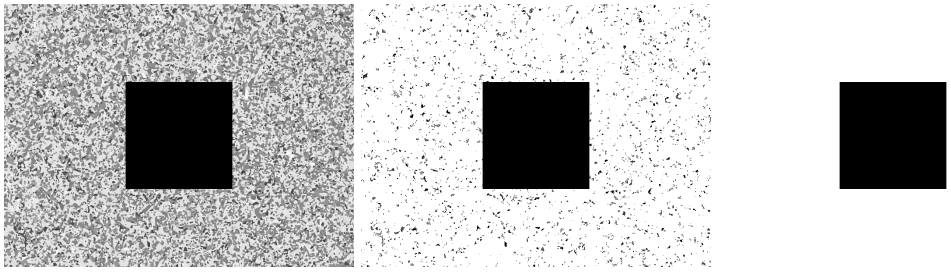
Um die Position oder den Winkel eines ICs oder Referenzpunktes zu bestimmen ist es notwendig, dessen Außenkanten zu erkennen. Dafür wurde die Funktion *getBorders()* entwickelt (siehe Listing E.2). Diese erkennt auf einem Bild das größtmögliche Objekt und gibt ein Rechteck als *RotatedRect* Datentyp zurück, welches das Objekt umschließt. Da ICs ebenfalls eine rechteckige Form besitzen, entspricht das gebildete *RotatedRect* der Außenkante des ICs. Bei runden Objekt gäbe es diesen Vorteil nicht, dort würde das Rechteck um das Objekt gelegt werden. Eine Voraussetzung für eine erfolgreiche Erkennung ist, dass das zu erkennende Rechteck die größte Fläche im Vergleich zu eventuellen anderen Objekten auf dem Bild einnimmt. Dies liegt an dem Aufbau der Funktion, welche nur das größte erkannte Objekt von einem *RotatedRect* umhüllt. Ist dies aufgrund von störenden Elementen an den Rändern des Bildes nicht der Fall, kann dies mit der in Kapitel 4.7 beschriebenen Methode kompensiert werden.

Die einzelnen Schritte der *getBorders()* Funktion sind im Folgenden beschrieben, in Abb. 4.5 ist zusätzlich das Ablaufdiagramm abgebildet.

1. Umwandeln in ein Graustufenbild:
   Als Vorbereitung auf die Binarisierung wird das original Farbbild (siehe Abb. 4.6 a) in ein Graustufenbild umgewandelt (siehe Abb. 4.6 b). Dabei werden die drei Farbkanäle Rot, Grün und Blau in einen Farbkanal umgewandelt. Dieser enthält nur Grauabstufungen zwischen Weiß und Schwarz.

2. Binarisierung des Bildes:
   Das Graustufenbild wird in ein binäres schwarz-weiß Bild umgewandelt (siehe Abb. 4.6 c). Es enthält dadurch nur noch zwei Farben, Schwarz und Weiß. Der entstandene hohe Kontrast wird für die hierauf folgenden Kontursuche benötigt. Die Binarisierung erfolgt mithilfe eines Schwellwertverfahren [5, S.135-138] und einer adaptiven Schwellwertbestimmung nach Otsu [22]. Das adaptive Verfahren hat den Vorteil, dass der Schwellwert für jedes Bild neu bestimmt wird und damit ein manuelles Festlegen des Wertes nicht notwendig ist. Ein fester Schwellwert hätte zusätzlich den Nachteil, dass er nur für die Umstände gelten würde, unter denen er ermittelt wurde. Bei unterschiedlichen Lichtverhältnissen könnte es zu unerwünschten Ergebnissen kommen, zum Beispiel zu einer Überbelichtung . Bei verrauschten Bildern und Bildern mit

ungleichmäßigen Hintergründen ist die Erkennung einer Kontur aufgrund der vielen kontrastreichen Stellen, welche nicht zu dem gesuchten Element gehören, nicht immer erfolgreich. Durch die Binarisierung werden diese Störungen entfernt (siehe Abb. 4.4), insofern ein passender Schwellwert gewählt wurde. Das funktioniert jedoch nur, wenn der Hintergrund grundsätzlich einen hohen Kontrast zu dem gesuchten Element hat. Für den Testhandler wurde dafür ein weißer Hintergrund über dem Saugheber befestigt. Die von OpenCV für die Binarisierung zur Verfügung gestellte Funktion ist die `threshold` Funktion.



(a) Original Bild (b) Falscher Schwellwert (c) Richtiger Schwellwert

Abbildung 4.4.: Auswirkung der Binarisierung auf einen ungleichmäßigen Hintergrund.

3. Alle Konturen erkennen:
   Nachdem ein binäres Bild mit daraus resultierenden, hohen Kontrast berechnet wurde, können daraus die Außenkanten der Elemente erkannt werden (siehe Abb. 4.6 d). OpenCV bietet dafür die `findContours` Methode an, welche zur Erkennung der Konturen einen Algorithmus von Suzuki und Abe [23] nutzt. Die Funktion wurde so parametrisiert, dass nur die äußersten Konturen eines Elementes erkannt werden. Die Konturen werden als Arrays von Koordinaten, folglich in Form eines Vektors, zurückgegeben. In dem einfachen Fall, dass die Kontur eines Rechteckes erkannt wurde, besteht das Array aus den vier Eckpunkt-Koordinaten.

4. Die Kontur der größten Fläche bestimmen:
   Zu diesem Zeitpunkt sind die Konturen aller erkannten Elemente als Koordinaten Arrays bekannt. Mithilfe der `contourArea` Funktion und einer Schleife, welche alle Konturen durchläuft, wird jene Kontur ermittelt, die die größte Fläche umspannt. (siehe Abb. 4.6 e).

5. Rechteck um die größte Fläche legen:
   Um die größte Kontur wird, mithilfe der `minAreaRect` Funktion, ein kleinstmögliches Rechteck gelegt. Dabei kann dieses Rechteck auch rotiert sein. Da es sich bei den hier zu erkennenden Elementen immer um Rechtecke (ICs, Referenzpunkte) handelt, stimmen die Außenkanten der Elemente mit dem erzeugten Rechteck überein. Weil

das erzeugte Rechteck vom Datentyp `RotatedRect` ist, sind Eigenschaften des Rechteckes wie Größe oder Winkel als Klassenattribute bekannt.

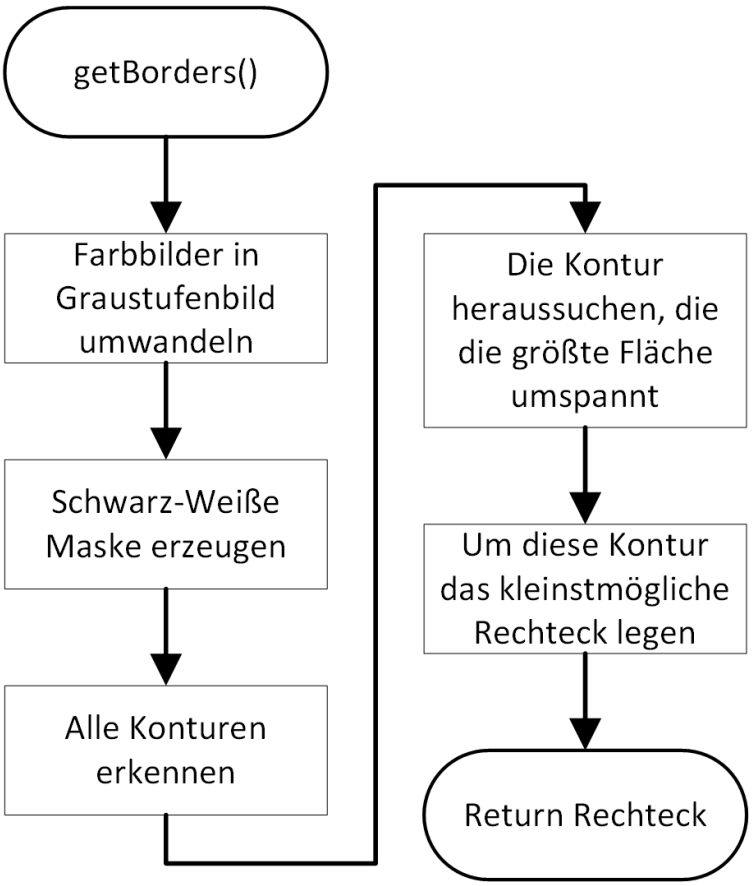
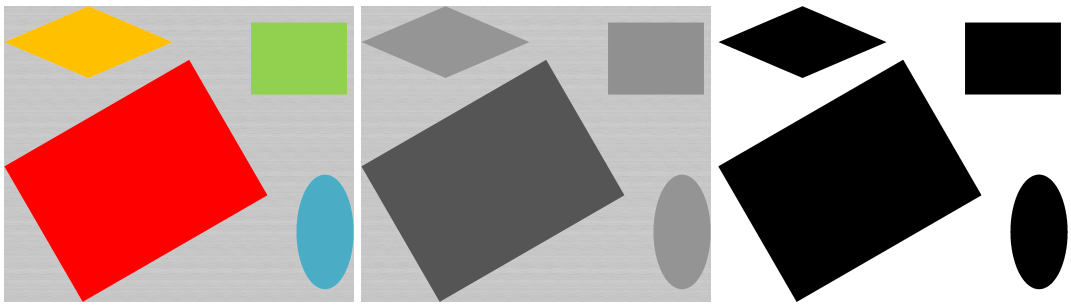
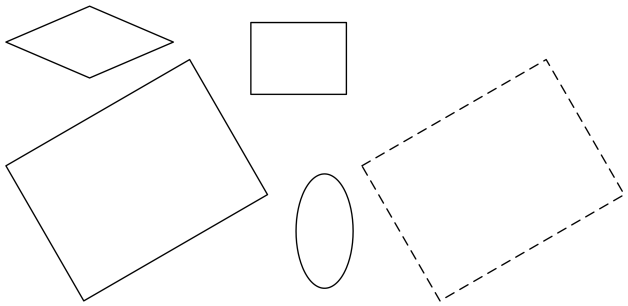


Abbildung 4.5.: Ablauf der Kantenerkennung



(a) Original Bild (b) Graustufenbild (c) Schwarz-weißes Binärbild



(d) Alle Konturen (e) Kontur der größten Fläche

Abbildung 4.6.: Ablauf der *getBorders()* Funktion (siehe Listing E.2)

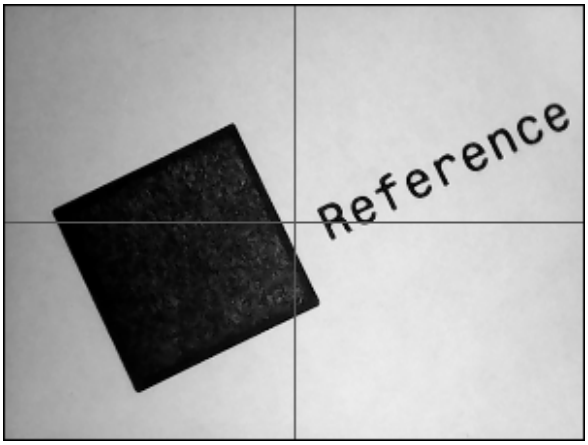
4.4. Ausgleich verdrehter Kameras

Da die verwendeten Kameras aus baulichen Gründen nicht exakt winklig zu den Achsen des LitePlacer ausgerichtet werden können, erfolgt eine softwareseitige Korrektur. Die Gründe für die Notwendigkeit unterscheidet sich bei den beiden Kameras wie folgt:

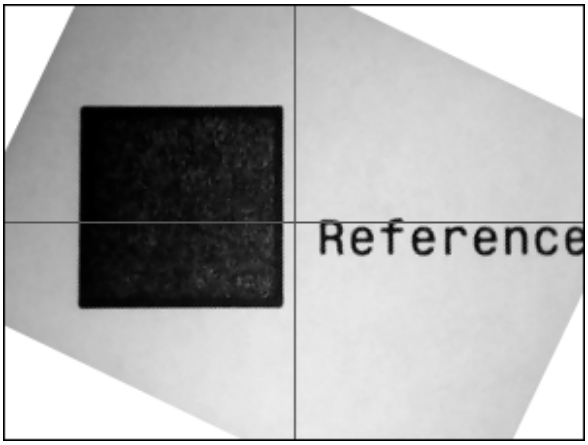
- Fahrtkamera:
  Die Fahrtkamera dient hauptsächlich zur Erkennung der Referenzpunkte. Für diese Funktion wäre eine exakte Ausrichtung in der X- und Y-Achse nicht nötig, da der Mittelpunkt eine verdrehten Quadrates der gleiche ist wie der eines geraden. Jedoch wird die Bildwiedergabe in der Benutzeroberfläche benutzt, um den Testhandler manuell zu bewegen und präzise zu positionieren. Wenn das Bild der Kamera nicht exakt parallel zu der X- und Y-Achse ist wirken alle Aufbauten, die in Wirklichkeit winkelig sind, in dem Bild verdreht. Wenn dann nur eine Achse verfahren wird, entweder die X- oder Y-Achse, sieht es auf dem Bild aus, als würde sich der Testhandler nicht gerade, sondern schief bewegen. Dadurch erschwert sich für den Benutzer die manuelle Steuerung.

- Tischkamera:
  Da die Tischkamera benutzt wird, um die ICs so auszurichten, dass sie in den Testsockel und die Trays passen, muss das Bilder dieser Kamera parallel zu den Trays und dem Testsockel sein, um eine einfache Ausrichtung zu ermöglichen. Wenn das Bild der Kamera winklig zu Tray und Sockel ist, hat ein IC, der rechtwinklig zu dem Bild ausgerichtet ist, auch den gleichen Winkel und passt somit winklig in Tray und Sockel.

Um die Bilder zu rotieren ist ein Bezugswinkel notwendig. Dieser liegt rechtwinkelig zu der X- und Y-Achse. Für die Fahrtkamera wird das untere Referenzquadrat des Failtray verwendet. Für die Tischkamera eine kleine Schablone mit einem Referenzquadrat verwendet, welche über die Tischkamera mit Anschlag an die Achsen gelegt werden kann um den Winkel des Quadrates zu bestimmen (siehe Abschnitt B.2.1). Die Kompensation der verdrehten Kameras lässt sich über die Funktion `on_calCam1pushButton_clicked`, für die Fahrtkamera, und `on_calCam2pushButton_clicked`, für die Tischkamera, durchführen (siehe Listing E.8). Beide Kameras müssen vor dem Start der Testroutine mithilfe dieser Funktionen kalibriert werden.



(a) Vor Winkel Korrektur



(b) Nach Winkel Korrektur

Abbildung 4.7.: Ausgleich einer verdreht eingebauten Kamera

4.5. Referenzierung im Raum

Um die Trays präzise anfahren zu können, muss der Testhandler die exakten Koordinaten des Trays kennen. Dafür wurden Trayhalterungen gefertigt (siehe Abb. 4.15 und Abschnitt B.2.1), in denen optische Marker integriert sind. Dessen genauer Abstand zu den Tray-Außenkanten ist bekannt. Diese Halterungen sind auf der Arbeitsfläche befestigt und rechtwinklig zu der X-Achse ausgerichtet. Dadurch, dass die Abstände zwischen Marker und Tray-Außenkanten bekannt sind, ist ein genaues Positionieren innerhalb des Trays möglich, nachdem die Fahrtkamera sich auf den Markern zentriert hat. Durch die kurze Differenz zwischen Tray und Marker (bei dem Failtray 30 mm, bei dem Source- und Passtray 15mm), werden mecha-

nisch begründete Abweichungen der verfahrenen Strecke gering gehalten. Vor der IC-Test Routine erfolgt ein automatisches Einmessen der drei Trays, wodurch kleine Änderungen der Lage der Trays kompensiert werden. Der komplette Ablauf der automatischen Kalibrierung und Referenzierung ist in Kapitel 7.1 beschrieben.

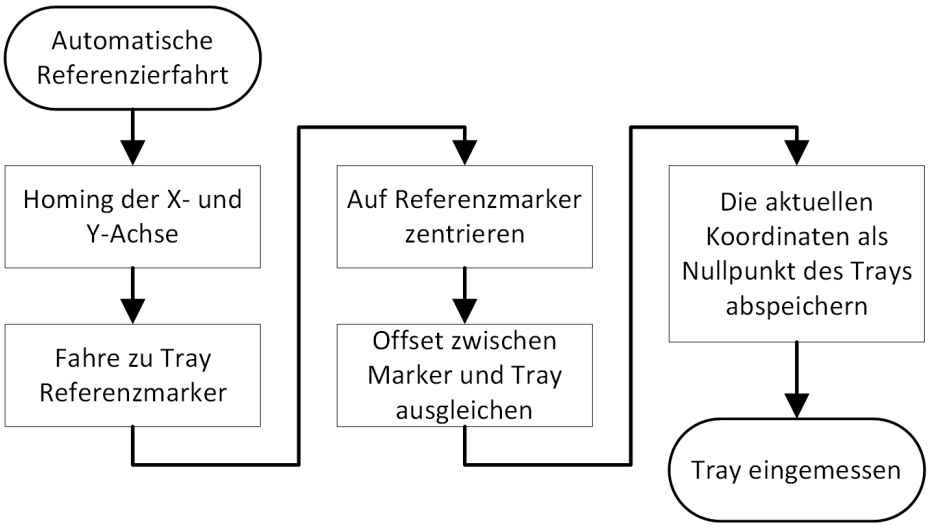


Abbildung 4.8.: Referenzieren eines Trays

4.5.1. Kompensation von Winkelungenauigkeiten des LitePlacers

Bei der Berechnung von Koordinaten wird von einem idealen, kartesischen Koordinatensystem ausgegangen. Das heißt, X- und Y-Achse stehen im 90° Winkel zueinander. In der Praxis stehen die X- und Y-Achse des LitePlacers jedoch aufgrund mechanischer Ungenauigkeiten nicht rechtwinklig zueinander, sondern im Winkel α (siehe Abb. 4.11). Dabei ist zu beachten, dass die linke Kante des LitePlacers als Anschlag für die genutzten Aufbauten fungiert. Alle drei Trays sowie der Testsockel befinden sich im rechten Winkel zu diesem Anschlag. Das hat zu Folge, dass ein Bewegen der Y-Achse eine Veränderung des Abstands zwischen Anschlag und aktueller X-Position bedingt, obwohl sich der Motor der X-Achse laut Software nicht bewegt hat. In Abbildung 4.12 a ist dargestellt, dass zwischen den beiden Punkte *x1* und *x2* eine Differenz in X-Richtung entsteht, wenn nur die Y-Achse verfahren wird. In einem ideal rechtwinkligen System gäbe es diese Differenz nicht. Da der Winkel α sich mit mechanischem Mitteln nicht exakt auf 90° einstellen lassen kann, erfolgt eine Kompensation in Software. Zum Ausgleich dieser Differenz wird die X-Achse verfahren (siehe Abb. 4.12 b), sodass *x1* und *x2* die selbe Differenz zum linken Anschlag haben. Um diese Kompensationsdifferenz zu berechnen, muss das Verhältnis zwischen verfahrener Y-Strecke und entstandenem X-Versatz bekannt sein. Dafür sind auf der Schablonen des Failtrays zwei

Referenzquadrate eingearbeitet (siehe Abschnitt B.2.1), die beide den gleichen Abstand *a* zu dem Anschlag haben (siehe Abb. 4.11). Der Abstand in Y-Richtung, *b*, ist konstruktionsbedingt ebenfalls bekannt. Diese beiden Marker werden angefahren und ihr Mittelpunkt gemessen. dafür wird die Funktion `goToCenterCam1` genutzt (siehe Listing E.10). Bei einem rechtwinkligem System würde bei beiden Markern die selben X-Koordinaten gemessen werden. Bei einem schiefen System ergäbe sich eine Differenz. Die X- und Y-Differenz beider Marker wird berechnet und das Verhältnis aus ihnen gebildet (siehe Abb. 4.9).

$$X_{offset} = X_{topleft} - X_{bottomleft} \tag{4.1}$$

$$Y_{offset} = Y_{topleft} - Y_{bottomleft} \tag{4.2}$$

$$Korrekturfaktor\ f_{ycor} = \frac{X_{offset}}{Y_{offset}} \tag{4.3}$$

Abbildung 4.9.: Berechnung des Korrekturfaktors f_{ycor}

Um den Korrekturfaktor zu verwenden muss dieser mit der zu verfahrenden Y-Strecke multipliziert werden. Das Ergebnis ist die zu kompensierende X-Strecke (siehe Abb. 4.10). In der Software wird diese Winkelkompensation durch die Funktion `measureYCompansation` durchgeführt (siehe Listing E.10).

$$X_{comp} = Y_{Strecke} \cdot f_{ycor} \tag{4.4}$$

Abbildung 4.10.: Verwendung des Korrekturfaktors *f*

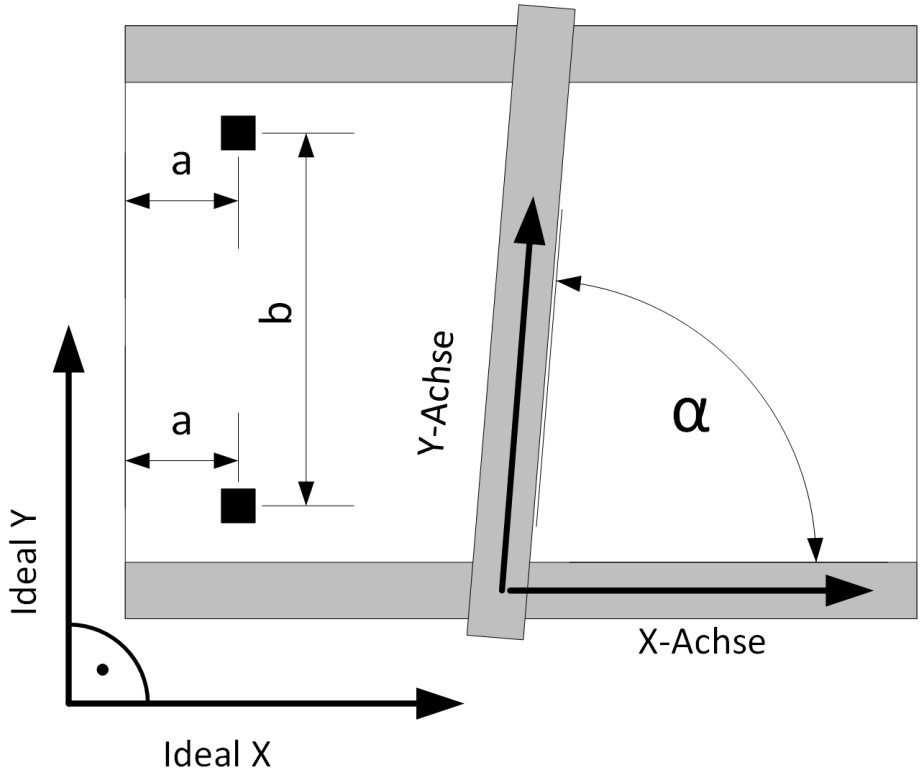
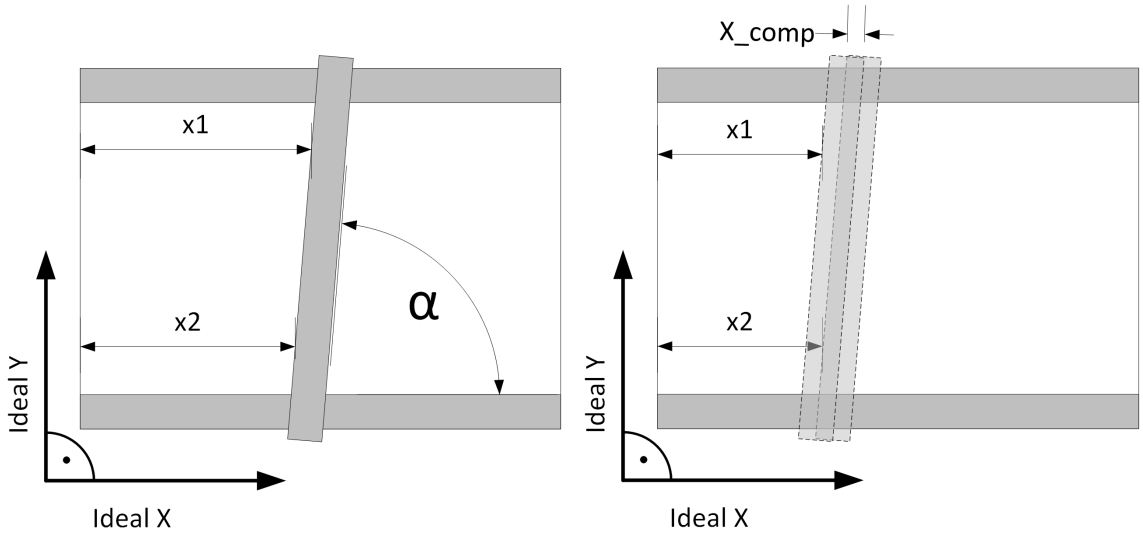


Abbildung 4.11.: Schematische Draufsicht auf LitePlacer mit stark überzeichnetem Winkel α



(a) Ohne Kompensation x1 $\neq$ x2 (b) Mit Kompensation x1 = x2

Abbildung 4.12.: Kompensation der nicht rechtwinkligen Y-Achse

4.5.2. Projektion der Pixel auf Längenkoordinaten

Digitale Bilder bestehen aus Pixeln welche keine Größeneinheit besitzen. Es ist daher zum Beispiel nicht möglich festzustellen, wie lang ein 400 Pixel breites Rechteck in Metern wäre. Dazu muss ein Faktor bekannt sein, der angibt, wie viele Pixel einer Längeneinheit entsprechen. Um diesen Faktor zu ermitteln muss die Größe des fotografierten Objekts bekannt sein. Daraus lässt sich das Verhältnis aus der Größe in zum Beispiel Zentimeter und Pixel bilden. Da die Größe einer Abbildung von dem Abstand der Kamera zu dem Motiv abhängt, gilt der berechnete Faktor nur für den beim Ermitteln eingestellten Abstand. In Abb. 4.13 entspricht *X* der bekannten Länge des Testobjektes. *Z* ist der Abstand zwischen Kamera und Testobjekt. Formel 4.5 beschreibt den Zusammenhang von gemessenen Pixeln und der realen Größe.

$$\frac{Pixel}{X[mm]} = Faktor \tag{4.5}$$

Als Beispiel sei *X=20 mm*, was auf dem Bild *200 Pixeln* entspricht. Dann wäre der Faktor:

$$\frac{200Pixel}{20mm} = 10\frac{Pixel}{mm} \tag{4.6}$$

Wird jetzt mit dem gleichen Abstand *Z* eine Länge von *125 Pixeln* gemessen, ist eine Umrechnung in Millimeter möglich:

$$\frac{125}{10} = 12.5mm \tag{4.7}$$

Für die Kalibrierung des Testhandlers werden Quadrate mit einer Kantenlänge von 20 mm verwendet. Diese werden erkannt und in Vektoren umgewandelt. Danach wird überprüft, ob alle Kanten die gleiche Länge haben, ob es also wirklich ein Quadrat ist. Wenn es als eins erkannt wurde, wird ein Kantenlänge in Pixeln mit den 20 mm verrechnet. Dabei kann die Referenzlänge mittels der Hardware Konfigurationsdatei eingestellt werden. Der Faktor ist für den Ausgleich von des Versatzes in X- und Y-Richtung notwendig (siehe Kapitel 4.5.4). Diese Kalibrierung muss für beide Kameras erfolgen, da sie in sich Auflösung und Abstand zum Motiv unterscheiden. Die Kalibrierung erfolgt mittels der Funktion `pixelToMillimeter` (siehe Listing E.2).

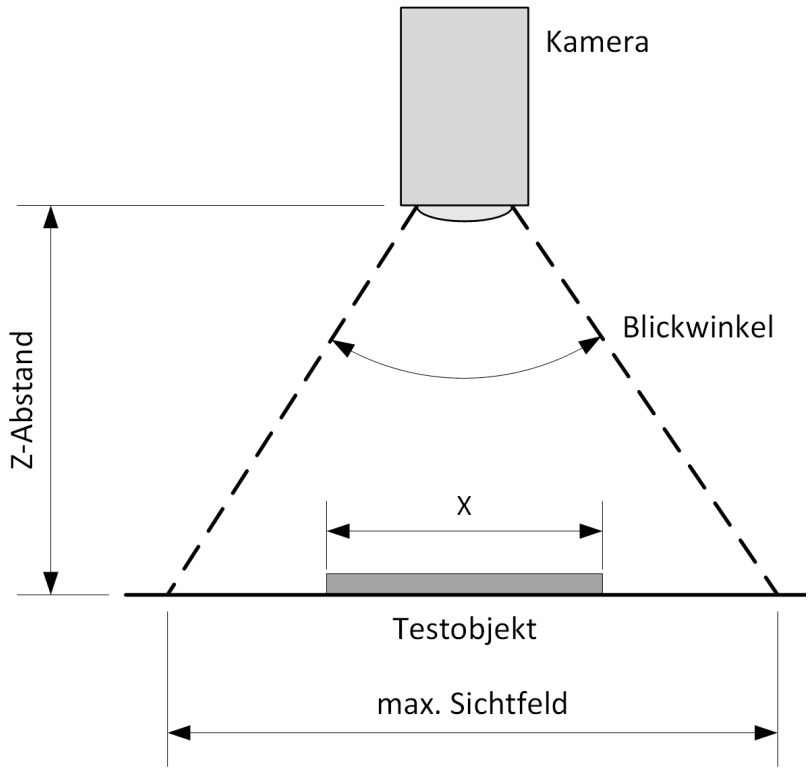


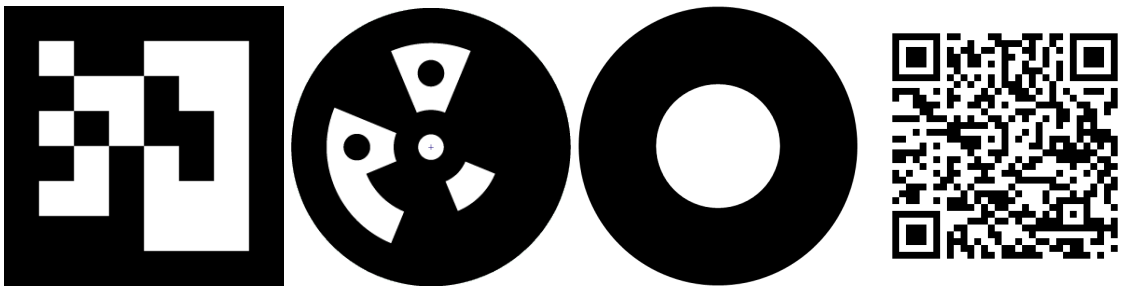
Abbildung 4.13.: Erkennung der verfügbaren Pixel pro Millimeter

4.5.3. Optische Referenzmarkierung

Eine optische Referenzmarkierung ist eine für Kameras gut erkennbare Markierung, die häufig für optische Positionserkennungen oder Messungen eingesetzt werden.
Häufige Anwendungen finden sich in:

- Film- und Computerspielindustrie für Motion Capturing
- Fertigung von elektronischen Platinen. Die Marker dienen zur Positionierung der Bestückungsautomaten um eine präzise Bestückung zu ermöglichen
- Messtechnik, zur Markierung relevanter Punkte
- Robotik, zur Referenzierung des Roboters im Raum

Um von einer Kamera gut erkannt werden zu können, muss ein optischer Marker einen hohen Kontrast zu seiner Umgebung aufweisen und ein vorgegebenes Muster besitzen, das durch die Bilderkennung erkannt wird. Idealerweise funktioniert die Erkennung in unterschiedlichen Beleuchtungen sowie bei verzerrten oder verschwommenen Bildern. Man kann zwischen aktiven und passiven Markern unterscheiden. Aktive Marker können zum Beispiel leuchten, wozu Energie benötigt wird. Passive Marker benötigen keine Energie, sie sind zumeist Aufkleber oder in Objekte integrierte Markierungen. Die Passiven Marker Typen unterscheiden sich zusätzlich noch darin, ob der individuellen Marker eine Information enthält, die erkannt werden und damit zugeordnet werden kann, oder ob alle Marker komplett gleich aussehen und dadurch nicht zwischen verschiedenen Positionen anhand der Marker unterschieden werden kann. Um den hohen Kontrast zu erreichen, sind die meisten Marker in schwarz-weiß gehalten und mit scharfen Kanten gezeichnet [2]. Es werden verschiedene Marker Varianten verwendet, von denen in Abb. 4.14 einige Beispiele zu sehen sind. Die Typen (a), (b) und (C) enthalten zusätzliche Informationen.



(a) *ARTag [6]* (b) *Intersense [12]* (c) Platinen Passermarke (d) QR-Code [9]

Abbildung 4.14.: Verschieden Arten von, zur optischen Referenzierung genutzten, Mustern. Muster (a) und (b) sind speziell für die Verwendung für die optische Erkennung entwickelt worden. . Muster (c) zeigt ein Marker, wie er häufig in der Fertigung von Platinen zur Referenzierung der Bestückungsautomaten genutzt wird. Muster (d) zeigt einen QR-Code. Ursprünglich nicht als optischer Referenzpunkt entwickelt, wird dieser dennoch dafür genutzt. Typ (a), (b) und (c) können zusätzliche Informationen enthalten.

Da durch die Entwicklung zur Erkennung der ICs bereits eine Funktion zur Erkennung von Rechtecken zu Verfügung stand (siehe Kapitel 4.3), wurden schwarze Quadrate auf weißem Hintergrund als Marker benutzt (siehe Abb. 4.15). Mit Hilfe der in Kapitel 4.5.4 beschriebenen Methoden wird die Fahrtkamera über dem Referenzquadrat zentriert und befindet sich dadurch an einer bekannten Position von der aus verfahren werden kann.

Abbildung 4.15.: Tray Halterung mit drei integrierten, quadratischen Referenzmarkern

4.5.4. Ausgleich des Versatzes in X- und Y-Dimension

Nachdem der Referenzmarker erkannt wurde, muss sich der Testhandler mittig über diesem positionieren. Dadurch können, ausgehend von der referenzierten Position, umliegende Positionen relativ angefahren werden, solange dessen Abstände zu dem Marker bekannt sind. Zum Beispiel ist die untere linke Ecke des Passtrays konstruktionsbedingt genau 15 mm in X-Richtung und -15 mm in Y-Richtung von dem Mittelpunkt des unteren linken Markers des Passtrays entfernt (siehe Abb. 4.15). Der Vorteil dieses Verfahren ist, dass nur die ungefähre Position des Markers bekannt sein muss. Sobald dieser im Sichtfeld der Kamera ist, kann die automatische Zentrierung erfolgen. Dadurch entfällt zum Beispiel ein manuelles Einmessen nach einer Änderung der Traybefestigung, solange sich der Marker noch im Blickfeld der Kamera befindet. Die Zentrierroutine der Fahrtkamera wurde in der Software durch die Funktion `goToCenterCam1` implementiert (siehe Listing E.10). Diese Funktion wird innerhalb einer Schleife aufgerufen, bis ein Schwellwert der X- und Y-Abweichung unterschritten wurde. Der Ablauf ist in Abb. 4.16 wiedergegeben und im Folgenden beschrieben:

1. Aktuelles Bild von Marker aufnehmen.
2. Das Bild um den Korrekturwinkel rotieren (siehe Kapitel 4.4).
3. Erzeugung eines gleichmäßigen Hintergrundes (siehe Kapitel 4.7).
4. Die Außenkanten des quadratischen Markers erkennen (siehe Kapitel 4.3).
5. Es wird die Differenz in X- und Y-Richtung zwischen dem Mittelpunkt des Bildes, also der Fahrtkamera, und dem Mittelpunkt des erkannten Markers bestimmt. Da diese in Pixel angegeben sind, erfolgt eine Umrechnung in Millimeter (siehe Kapitel 4.5.2).
6. Der berechnete Versatz wird mit einem Schwellwert verglichen, der die maximal erlaubten Abweichung festlegt. Dieser Wert beträgt 0.05 mm. Bei Schwellwerten die geringer waren, konnte die Schleife, welche die Funktion aufruft, nicht beendet werden. Die Präzision der Kantenerkennung in Zusammenspiel mit der Genauigkeit der mechanischen Umsetzung lassen keine genauere Positionierung zu. Liegt der Wert des Versatzes unter dem Schwellwert, ist die Fahrtkamera ausreichend mittig positioniert. Ist der Wert größer, wird dies durch das Verfahren der X- und Y-Achse ausgeglichen. In dem Fall wird nach dem Positionieren ein neues Bild aufgenommen und es wird erneut überprüft und ausgeglichen.

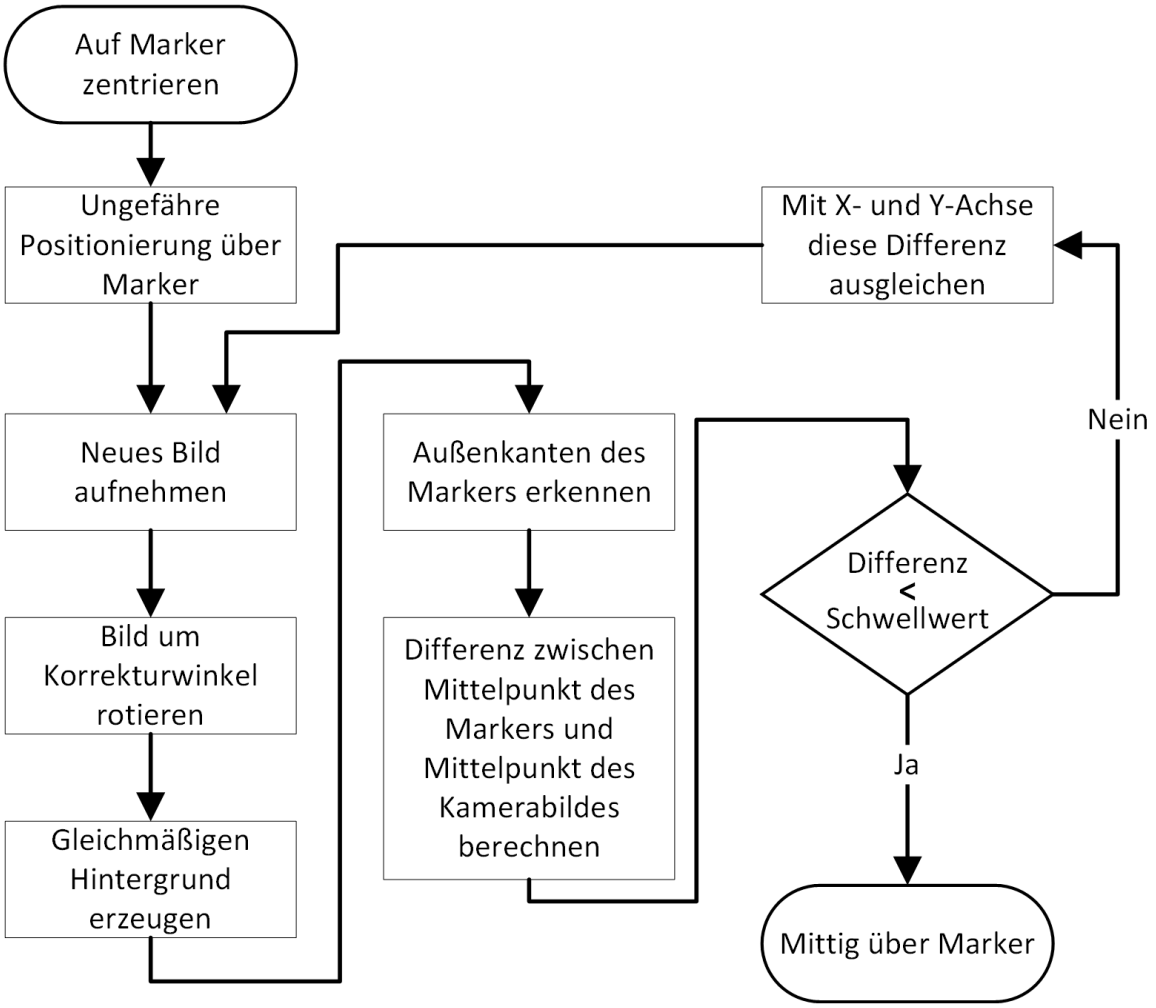


Abbildung 4.16.: Ablaufdiagramm für die Positionierung über einem optischen Marker

4.6. Orientierungs- und Positionserkennung des IC

Um den IC genau in den Testsockel und das Tray legen zu können, müssen folgende Punkte erfüllt sein:

- Der IC muss im gleich Winkel ausgerichtet sein wie der Testsockel beziehungsweise die Trays.
- Die Ausrichtung des ICs muss mit der des Testsockels, beziehungsweise der der Trays, übereinstimmen. Das heißt, der erste Pin des ICs muss mit dem ersten Pin des Testsockels übereinstimmen.

- Der Mittelpunkt des ICs muss bekannt sein, um den IC mittig in dem Testsockel beziehungsweise einem Tray zu positionieren.

Um diese Voraussetzungen zu erfüllen wurde eine automatisierte Mess- und Korrekturroutine entwickelt. Im Quellcode heißt sie `adjustChip` (siehe Listing E.10). Die Routine besteht aus mehreren, eigenständigen Funktionen, die in den nachfolgenden Kapiteln erläutert werden. Der Gesamtüberblick der Routine ist in Abb. 4.17 dargestellt. Der Ablauf wird im Folgenden beschrieben:

1. Saugheber mit IC über der Tischkamera positionieren.
2. Bild aufnehmen und zur Auswertung vorbereiten. Das bedeutet:
   - Bild um den Korrekturwinkel zu rotieren (siehe Kapitel 4.4).
   - Bild horizontal spiegeln, da die Tischkamera von unten sieht und damit zu der Draufsicht der Arbeitsfläche gespiegelt ist.
   - Erzeugung eines gleichmäßigen Hintergrundes (siehe Kapitel 4.7).
3. Überprüfen, ob der IC rechtwinklig zu den Achsen des Testhandlers ist und dadurch in den Testsockel und die Trays eingesetzt werden kann. Ist der IC verdreht, wird dies korrigiert (siehe Kapitel 4.6.2).
4. IC mittig über Kamera positionieren (siehe Kaptiel 4.6.1).
5. Von dem rechtwinklig ausgerichtetem IC eine neues Bild aufnehmen und wie unter Punkt 2 beschrieben vorbereiten.
6. Den Bildbereich, der den IC enthält, zuschneiden und auf die Größe der Mustervorlage skalieren (siehe Kapitel 4.6.4).
7. Den Ausschnitt des ICs mit der Mustervorlage vergleichen (siehe Kapitel 4.6.3).
8. Falls der Bildvergleich erkennt, dass der IC mit der Mustervorlage übereinstimmt, hat der IC die richtige Orientierung und kann in den Testsockel gesetzt werden.
9. Falls die Bilderkennung keine Übereinstimmung erkennt, wird der IC um 90° gedreht und die Routine ab Punkt 4 wiederholen. Dies wird bis zu achtmal wiederholt, was zwei komplette Umdrehungen des ICs bedeutet. Dadurch können während des zweiten Durchlaufs noch ICs erkannt werden, die beim ersten Durchlauf unerkannt blieben.
10. Wurde auch nach der achten Rotation keine Übereinstimmung gefunden, erfolgt eine Überprüfung, ob der Saugheber überhaupt einen IC angehoben hat. Eine Möglichkeit dafür kann sein, dass sich der IC im Sourcetray verhakt hat. Das Überprüfen erfolgt mittels Bilderkennung (siehe Kapitel 4.6.3).

11. Ist ein IC am Saugheber befestigt und es kann trotzdem keine Ausrichtung erkannt werden, wird der IC in das Failtray abgelegt.

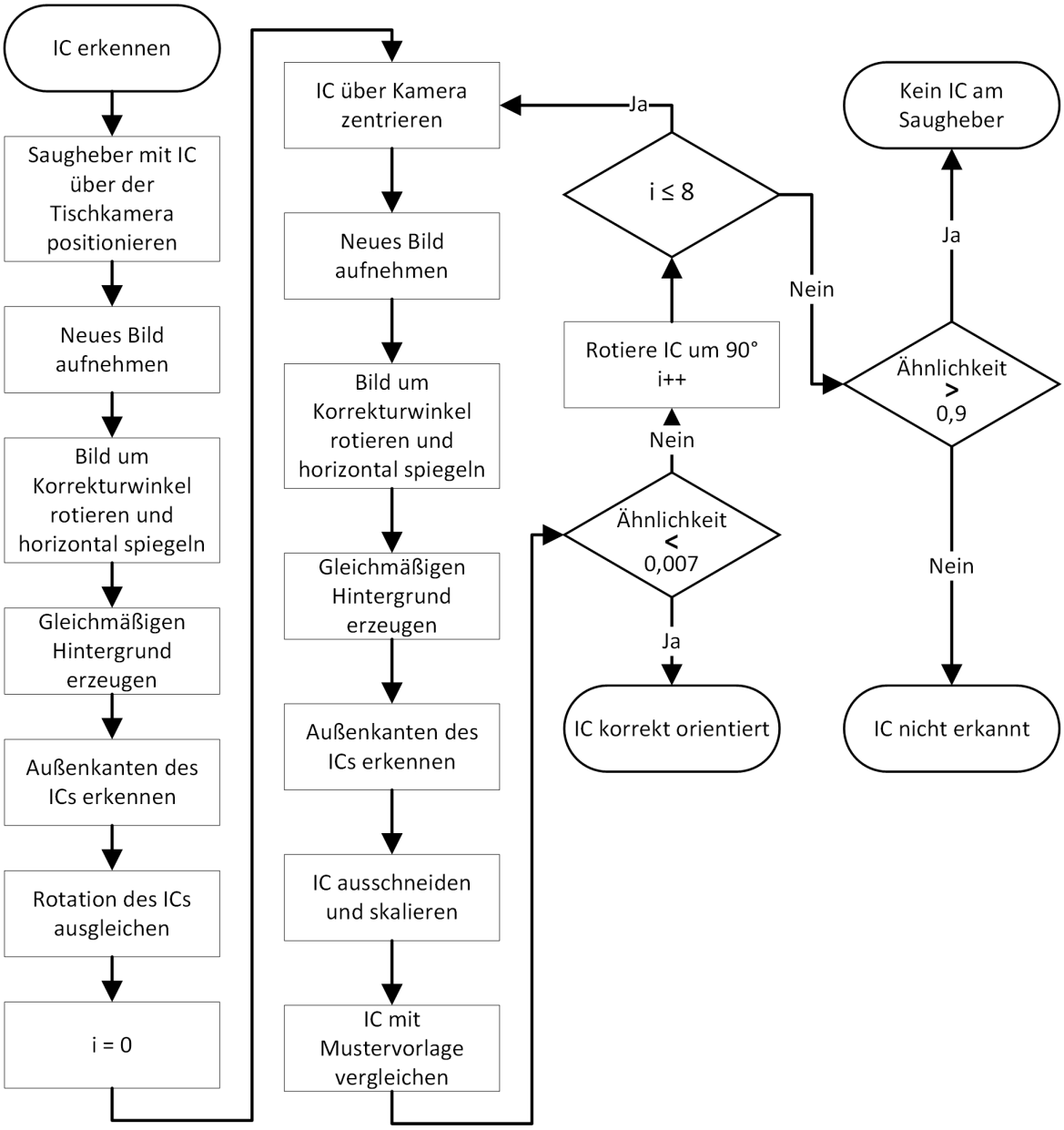


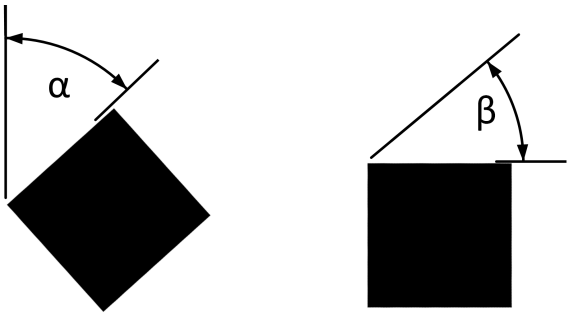
Abbildung 4.17.: Ablauf der Erkennung und Ausrichtung eines IC

4.6.1. Zentrieren des IC

Um den Mittelpunkt des ICs über dem Mittelpunkt der Kamera zu positionieren, wird die gleiche Funktion wie in Abschnitt 4.5.4 genutzt. Der einzige Unterschied ist, dass die Tischkamera verwendet wird und das Bild dementsprechend horizontal gespiegelt werden muss. Der Versatz der dabei in X- und Y-Richtung verfahren wird, wird gespeichert. Diese Werte dienen als Offset für die Positionierung im Testsockel und den Trays. In der Software ist die Zentrierung über der Tischkamera durch die Funktion `goToCenterCam2` implementiert (siehe Listing E.10). Sie wird durch eine Schleife solange aufgerufen, bis der Schwellwert der X- und Y-Abweichung unterschritten wird. Dieser ist wie im Abschnitt 4.5.4 0.05 mm.

4.6.2. Korrektur des Rotationswinkels

Da die Kontur des ICs als `RotatedRect` Datentyp vorliegt, kann der Winkel, um den das Rechteck rotiert ist, einfach als Membervariable `angle` ausgelesen werden. Aus diesem wird der Korrekturwinkel berechnet und die E-Achse um diesen rotiert (siehe Abb. 4.18).



(a) Verdrehter IC, erkannt wurde Winkel α

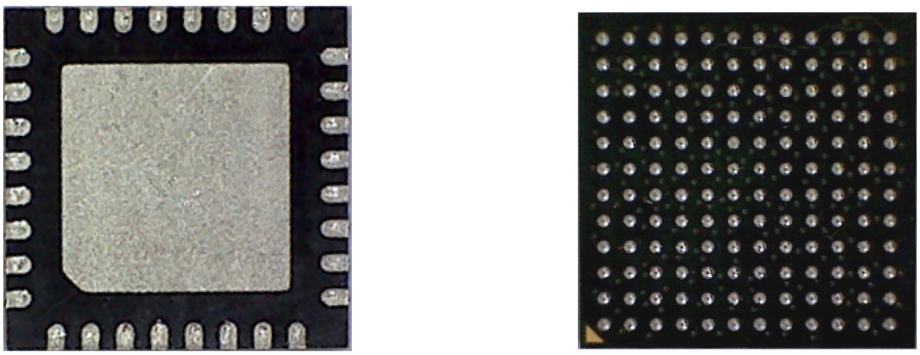
(b) Korrigierter IC, Korrekturwinkel β wurde berechnet

Abbildung 4.18.: Korrektur eines rotierten ICs

4.6.3. Vergleich von Bildern

Die ICs müssen in der korrekten Ausrichtung in den Testsockel gelegt werden, damit sich die richtigen Anschlüsse miteinander verbinden. Wird beispielsweise ein quadratischer IC, um 90° verdreht in einen Testsockel gesetzt, kann es sein, dass es zu der Zerstörung des ICs oder der Testelektronik kommt. Mögliche Ursachen hierfür wären die Verbindung der Stromversorgung mit empfindlichen Eingängen des ICs oder ein Kurzschluss der Testelektronik.

Um dies zu verhindern, müssen die ICs, bevor sie getestet werden können, auf ihre Ausrichtung überprüft und gegebenenfalls korrigiert werden. Da zur Überprüfung die Unterseite des ICs genutzt wird, muss diese Merkmale aufweisen, anhand derer eine Ausrichtung erkannt werden kann. Diese Merkmale unterscheiden sich je nach verwendeten Gehäusetyp. In Abbildung 4.19 sind zwei Beispiele davon abgebildet.



(a) QFN32 Gehäuse, abgeschrägte Ecke der Masse-fläche (unten links)

(b) BGA144 Gehäuse, goldenes Dreieck (unten links)

Abbildung 4.19.: Markierung bei verschiedenen Gehäusetypen

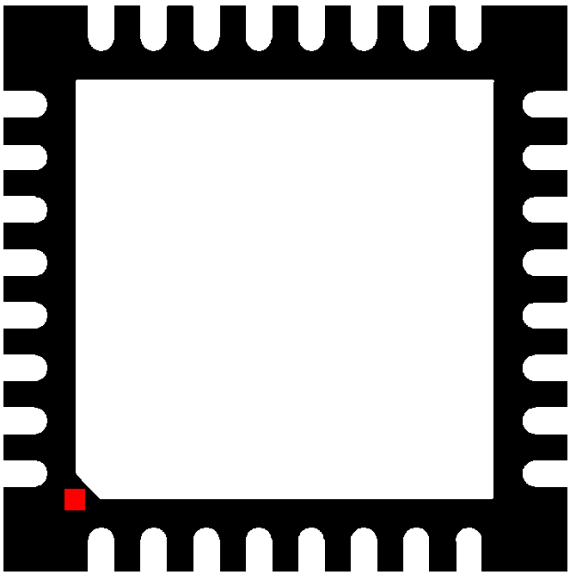


Abbildung 4.20.: Mustervorlage für einen QFN32 Gehäuse mit der markanten Stelle unten links. Rot eingefärbt ist der zu vergleichende Bereich

Um diese markante Stelle zu finden wird der ICs mit einer Mustervorlage (siehe Abb. 4.20) verglichen. Jedoch soll nur der IC mit der Vorlage verglichen werden, der Hintergrund stört dabei. Darum wird der Bildausschnitt, der den IC enthält, mit der, in Kapitel 4.6.4 beschriebenen, Methode ausgeschnitten und skaliert. Wird erkannt, dass beide Bilder gleich sind, entspricht die Ausrichtung des ICs der Mustervorlage. Von dieser ist die Ausrichtung bekannt. Da der Vergleich des ganzen Bildes sich als unzuverlässig herausgestellt hat, wird nur ein Ausschnitt verglichen, der die markante Stelle enthält. Der gewünschte Ausschnitt wird festgelegt, indem in der Vorlage das zu vergleichende Gebiet in Rot eingefärbt wird (siehe Abb. 4.20). Durch das Rot wird der Bereich automatisch erkannt und es müssen keine Koordinaten manuell ermittelt werden.

Um Bilder miteinander zu vergleichen, ist in der Bildverarbeitung eine pixelbasierte Kreuzkorrelation üblich [4, S 9-10]. Aus Gründen der Einfachheit wurde hier allerdings ein direkter, pixelweiser Vergleich gewählt. Dafür müssen beide Bilder folgende Voraussetzungen erfüllen:

- Beide Bilder müssen die selbe Höhe und Breite haben, damit jeder einzelne Pixel verglichen werden kann.
- Beide Bilder müssen binär in den Farben Schwarz und Weiß sein, um Pixel vergleichen zu können.

Der eigentliche Vergleich besteht aus drei Schritten:

1. Subtraktion beider Bilder:
   Beide Bilder werden voneinander elementweise Subtrahiert (siehe Abb. 4.21). Da beide Matrizen binär sind, also nur die Werte 1 (Weiß) und 0 (Schwarz) enthalten können, ergeben sich aus der Subtraktion zweier gleicher Pixel immer eine 0 und aus der Subtraktion zweier unterschiedlicher Pixel immer ± 1.

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} - \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11} - B_{11} & A_{12} - B_{12} \\ A_{21} - B_{21} & A_{22} - B_{22} \end{pmatrix} \tag{4.8}$$

Abbildung 4.21.: Subtraktion zweier Matrizen

2. Zählen der ungleichen Pixel:
   Je höher die Anzahl an ungleichen Pixeln ist, desto verschiedener sind die Matrizen. Bei zwei identischen Bilder würde die Ergebnismatrix nur Nullen enthalten, bei dem Vergleich eines Bildes mit seinem Negativ nur Einsen.

3. Normierung:
   Das Ergebnis wird normiert, indem das Verhältnis aus der Anzahl der ungleichen Pixel und der Anzahl von Elementen gebildet wird:

$$Aehnlichkeit\ \ s = \frac{ungleiche\ Pixel}{Reihen \cdot Spalten} \tag{4.9}$$

   Ein *s = 0* bedeutet maximale Ähnlichkeit, ein *s = 1* bedeutet maximalen Unterschied.

Im Folgenden sind zwei Beispiel aufgeführt

- Vergleich zweier identischer Matrizen:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

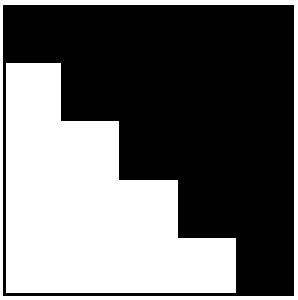


Abbildung 4.22.: Matrix A (Als Matrix und als grafisches Bild)

  Es erfolgt die Subtraktion $A - A$:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} - \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{4.10}$$

  Die Ergebnismatrix enthält nur Nullen, damit lässt sich die Ähnlichkeit *s* wie folgt berechnen:

$$s = \frac{0}{5 \cdot 5} = 0 \tag{4.11}$$

  Beide Bilder sind dementsprechend identisch.

- Vergleich zweier unterschiedlicher Matrizen:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

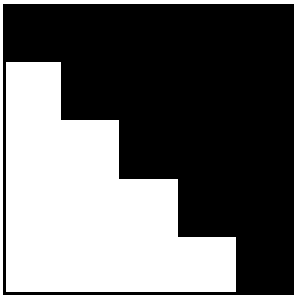


Abbildung 4.23.: Matrix A (Als Matrix und als grafisches Bild)

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

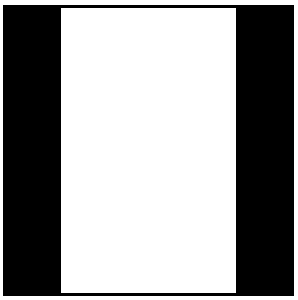


Abbildung 4.24.: Matrix B (Als Matrix und als grafisches Bild)

Es erfolgt die Subtraktion $A - B$:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -1 & -1 & -1 & 0 \\ 1 & -1 & -1 & -1 & 0 \\ 1 & 0 & -1 & -1 & 0 \\ 1 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{4.12}$$

Die Ergebnismatrix enthält 13 Elemente, die nicht Null sind. Damit lässt sich die Ähnlichkeit *s* wie folgt berechnen:

$$s = \frac{13}{5 \cdot 5} = 0,52 \tag{4.13}$$

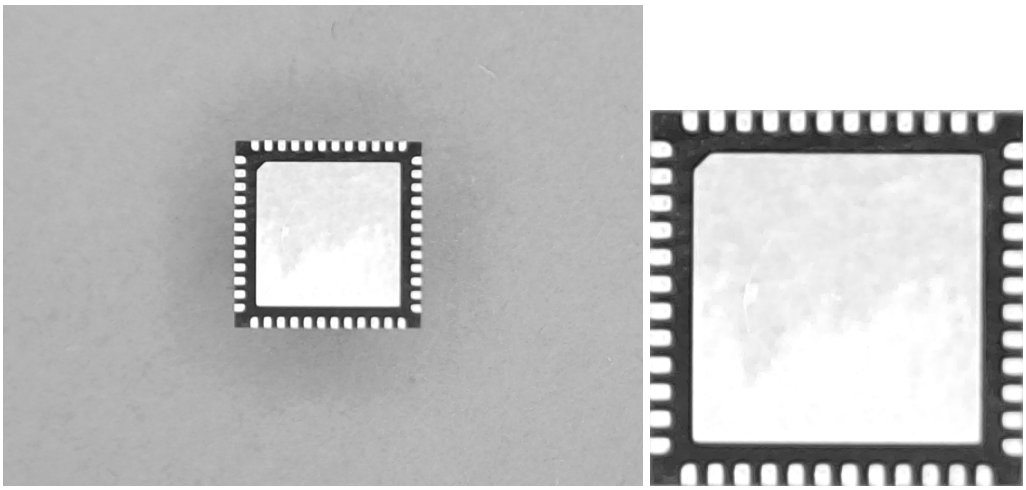
Ungefähr die Hälfte der Pixel sind gleich.

In der Realität können Kamerabilder, aufgrund von Umgebungseinflüssen und Eigenschaften der Kamera, niemals exakt der Mustervorlage gleichen. Deshalb ist ein Schwellwert notwendig, mit dem entschieden wird, welche Bilder noch als gleich zu erkennen sind und welche nicht. Im Verlauf der Entwicklung hat sich, durch das testen verschiedener Werte, ein Schwellwert von $s = 0.007$ als gut funktionierend herausgestellt. Zusätzlich ist es auch

möglich zu erkennen, ob der Saugheber überhaupt ein IC angehoben hat. In diesem Fall wird eine Ähnlichkeit größer $s = 0.9$ gemessen. Dieser Wert wurde ebenfalls durch Tests ermittelt. In der Software besteht der Bildvergleich aus zwei Funktionen. Zm einen aus der `compareMat` Funktion, welche den pixelweisen Vergleich von zwei gleichgroßen binarisierten Bildern durchführt und das Ergbnis normiert. Zum anderen aus der `compareImage` Funktion, welche ein Bild für den pixelweisen Vergleich vorbereitet (Binarisieren, Größe anpassen) und die zu vergleichende Fläche aus der Mustervorlage ausliest. Beide Funktionen sind im Listing E.2 wiedergegeben.

4.6.4. Ausschneiden und normieren des IC Bild

Um den IC mit der Mustervorlage vergleichen zu können ist es notwendig, dass aus dem Gesamtbild (siehe Abb. 4.25 a) der Bildausschnitt verwendet wird, der den IC beinhaltet (siehe Abb. 4.25 b). Zusätzlich muss dieser Ausschnitt die selbe Dimension wie die Mustervorlage besitzen, um mit ihr verglichen werden zu können. Das Ausschneiden des ICs funktioniert mit der in Kapitel 4.3 beschriebenen Kantenerkennung. Der Inhalt der erkannte Kontur wird dafür ausgeschnitten und in einem neuen `Mat` Objekt gespeichert. In der Software wird dafür die Funktion `cropChip` benutzt (siehe Listing E.2). Diese erzeugt zuerst einen einheitlichen Hintergrund (siehe Abschnitt 4.7), danach wird mit der `getBorders` Funktion der IC erkannt und ausgeschnitten. Für die Skalierung wird die OpenCV Funktion `resize` verwendet. Damit wird der Bildausschnitt auf die Maße der Mustervorlage skaliert. Diese wurde auf 1000 x 1000 Pixel festgelegt, um den mehrheitlich quadratischen Gehäuseformen zu entsprechen.



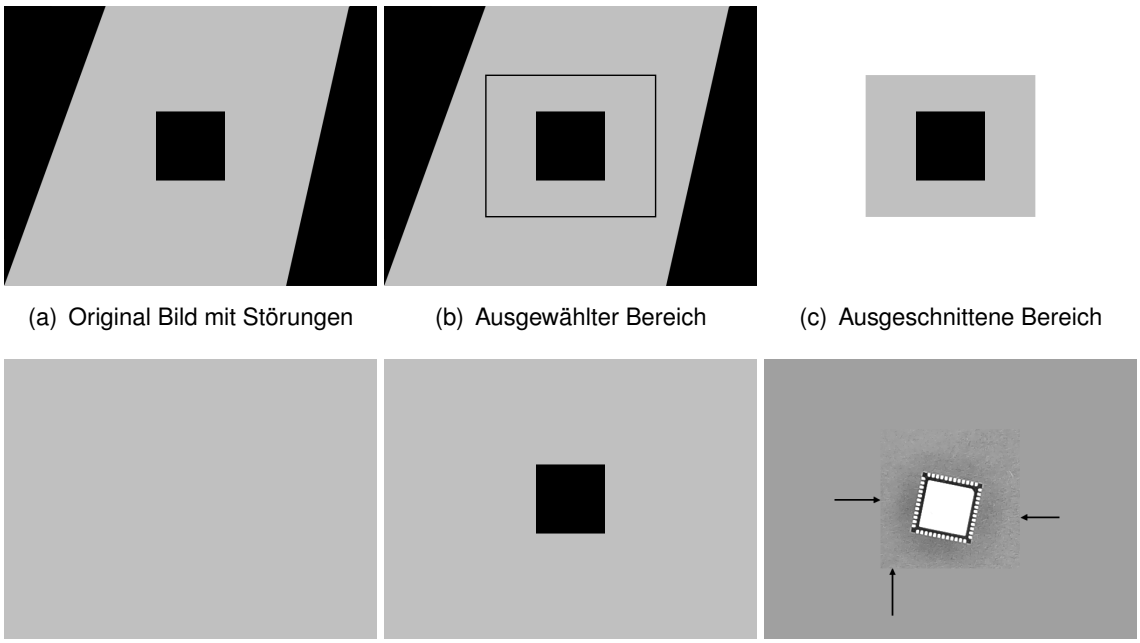
(a) Aufnahme der Tischkamera nach Winkelkorrektur (b) Ausgeschnitter IC

Abbildung 4.25.: Automatisches Erkennen und Ausschneiden des ICs.

4.7. Einfluss des Hintergrundes

Für die erfolgreiche Erkennung eines Objekts wird ein hoher Kontrast zwischen dem Objekt und der Umgebung, beziehungsweise dem Hintergrund benötigt. Das zu erkennende Objekt muss außerdem die größte Fläche des Bildes (abgesehen von dem Hintergrund) einnehmen. Wenn am Rand des Bildes, zum Beispiel durch Bauteile oder Schattenwurf, Flächen im Kontrast zum Hintergrund stehen, die größer als die Flächen des eigentlichen Objekts sind (siehe Abb. 4.26 a), so wird nicht das Objekt, sondern die Störung als gesuchte Fläche erkannt. Um das zu verhindern, wird der mittlere Teil des Bildes, der das zu erkennende Objekt enthält, ausgeschnitten (siehe Abb. 4.26 c). Die Größe des Ausschnitts (siehe Abb. 4.26 b) wird manuell festgelegt und wurde durch Tests ermittelt. Das Originalbild wird komplett in dem Farbton eingefärbt, der sich an der Schnittstelle befand (siehe Abb. 4.26 d), zum Beispiel an der oberen linken Ecke des Ausgewählten Bereichs. Danach wird der ausgeschnittene Bereich wieder in das original Bild eingefügt (siehe Abb. 4.26 e). Da der Hintergrund nicht an allen Stellen der Schnittkante den selben Farbton hat, ist die Schnittkante nach dem Wiedereinfügen durch leichte Farbänderungen zu erkennen. Eine Notwendigkeit die Kante verschwimmen zu lassen besteht aber nicht, da der Kontrast an der Kante sehr gering ist (siehe Abb. 4.26 f). In der Software erfolgt dies mit der `generateEvenBackround` Funktion (siehe Listing E.2).

Benötigt wird die Funktion bei den schwarzen Referenzmarkern des Source- und Passtrays, da die ebenfalls schwarzen Trays am Rand des Bildes zu sehen sind, wenn die Fahrtkamera mittig über den Markern positioniert ist.



(a) Original Bild mit Störungen (b) Ausgewählter Bereich (c) Ausgeschnittene Bereich

(d) Neu eingefärbter Hintergrund (e) Ausgeschnittene Bereich wieder eingefügt (f) Sichtbare Kanten nach dem Einfügen

Abbildung 4.26.: Entfernung von Störungen am Bildrand

4.8. Einfluss der Beleuchtung

Für die Erkennung der relevanten Details auf der Unterseite des ICs ist ein kontrastreiches und scharfes Bild nötig. Da die Tischkamera ist in einer lichtdichten Befestigung unterhalb des Tisches sitzt und beim Filmen des ICs durch den mittransportieren Hintergrund das Umgebungslicht weitestgehend abgeschirmt wird, ist eine gesonderte Beleuchtung notwendig. Die verwendete Mikroskopkamera besitzt einen eingebauten, einstellbaren LED Leuchtring. Allerdings strahlt dieser direkt auf den zu filmenden Gegenstand. Durch die metallenen Anschlüsse des ICs wird dieses direkte Licht stark reflektiert und erzeugt so eine Blendwirkung. Kontrastreiche und scharfe Bilder sind dadurch, mit der verwendeten Kamera, nicht möglich (siehe Abb. 4.28 a). Es wird daher eine indirekte Beleuchtung verwendet. Diese basiert darauf, dass es keine punktförmigen Lichtquellen gibt und das Licht über Reflexionen zu dem Ziel gelangt. Dadurch werden die direkten Reflexionen in die Kamera vermieden und es können die Metallflächen ohne Überbelichtung photographiert werden. Dafür wurde ein Ringlicht mit einem Diffusor zum streuen des Lichts entworfen und 3D gedruckt (siehe Abb. 4.27). Die Einzelheiten des Aufbaus der Beleuchtung sind im Anhang unter A.4 nachzulesen. Die Helligkeit der indirekten Lichtquelle muss zusätzlich einstellbar sein, um an die Umgebung an-

gepasst zu werden. In Abbildung 4.28 b ist zu sehen was eine zu helle indirekte Beleuchtung bewirkt. In Abbildung 4.28 c ist die Helligkeit auf ein gut nutzbares Maß eingestellt. Ausreichende Schärfe und Kontrast sind gegeben und jeder einzelne Anschluss lässt sich deutlich erkennen.

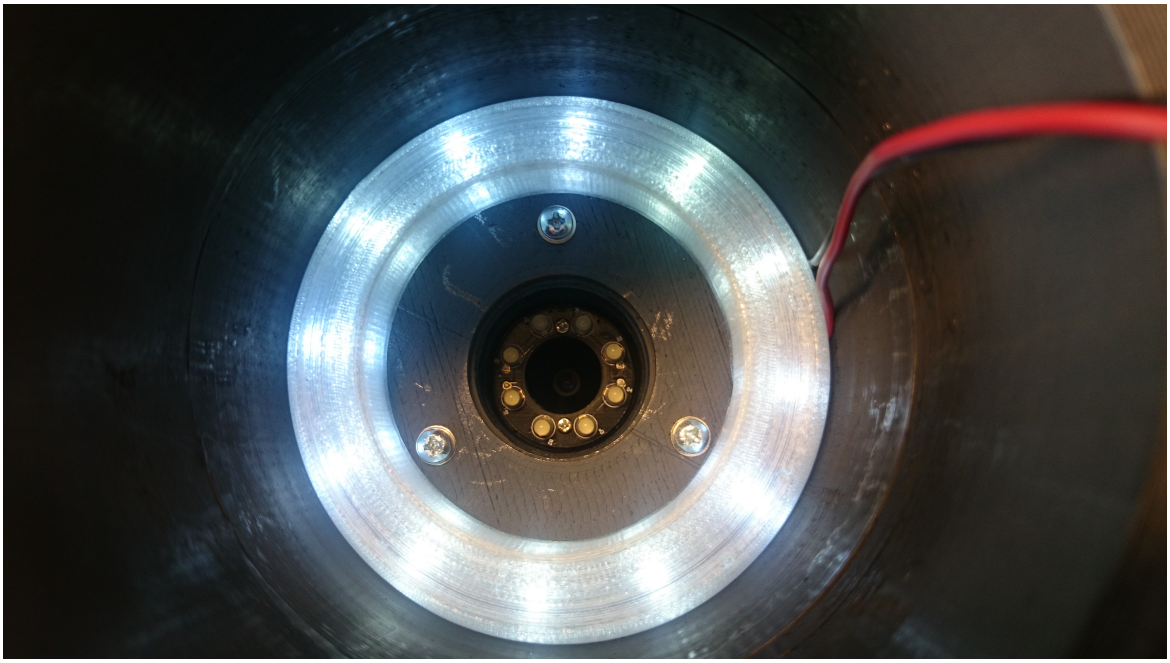
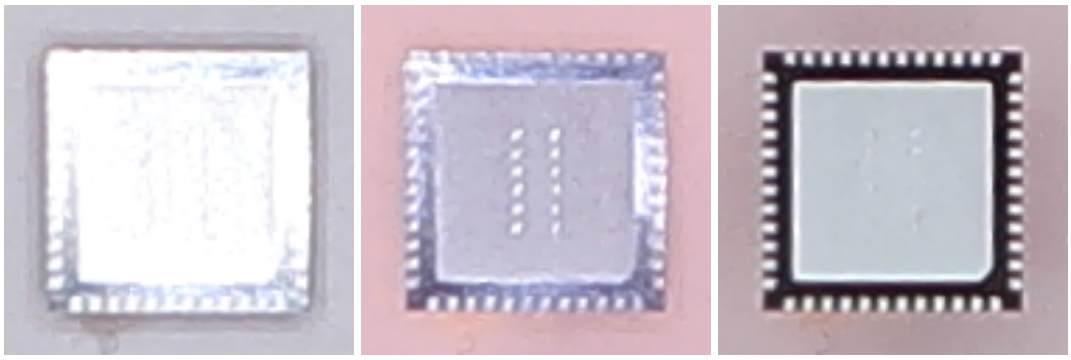


Abbildung 4.27.: Leuchtring der Tischkamera



(a) Direkte Beleuchtung (b) Zu helle indirekte Beleuchtung (c) Richtig eingestellte indirekte Beleuchtung

Abbildung 4.28.: Auswirkung von verschiedenen Beleuchtungsarten

5. Grafische Benutzeroberfläche

Für eine einfache Bedienung des Testhandlers war die Entwicklung einer graphischen Benutzeroberfläche[1] notwendig. Ziel war es, eine intuitiv zu bedienende Steuerkonsole mit optischem Feedback zu schaffen. Sie soll dem Benutzer die Möglichkeit geben, den Testhandler zu bedienen, ohne dass ein großes Hintergrundwissen benötigt wird. Durch die Vorgabe die Steuerung des Testhandlers in die TMCL-IDE zu integrieren, war die Nutzung des Qt-Frameworks naheliegend. Da die TMCL-IDE komplett auf diesem basiert, hat die Nutzung eine einfachere Integration und Wartung zur Folge. Im Folgenden soll eine kurze Übersicht über das Qt-Framework, sowie über das Bedienkonzept und die Benutzeroberfläche gegeben werden. Für weitere Informationen über Qt sei auf die Dokumentation unter [1] verwiesen.

5.1. Qt-Framework

Das Qt-Framework ist eine Ansammlung von Softwarebibliotheken für C++. Den größten Anteil davon haben die Bibliotheken zur Entwicklung graphischer Benutzeroberflächen. Zusammen mit dem *Qt Creator*, einer eigenen Entwicklungsumgebung von Qt, ermöglichen diese eine einfache Entwicklung von Benutzeroberflächen. Diese sind per Drag-and-Drop[2] aus vorhandenen Elementen, wie Schaltern oder Texteingaben, zusammenstellbar oder können als XML Datei angegeben werden. Neben den GUI-Bibliotheken enthält Qt viele Erweiterungen für C++, wie zum Beispiel Funktionen zum Laden und Speichern von Initialisierungsdateien oder die Auslagerung von Funktionen in eigene Threads. Qt wird seit 1995 entwickelt und ist plattformübergreifend. Für die Entwicklung dieser Bachelor Arbeit wurde die Qt Version 5.6 benutzt.

[1] Graphical User Interface, GUI

[2] Ziehen und Ablegen

5.2. Bedienkonzept

Damit der Testhandler in seinen Grundfunktionen steuerbar ist, waren folgende Bedienelemente eingeplant:

- Bildwiedergabe der Kameras:
  Die Bildwiedergabe ist für den eigentlichen Testablauf nicht notwendig, allerdings erleichtert sie dem Benutzer die manuelle Steuerung. Zusätzlich ermöglicht sie beim Einrichten und Kalibrieren eine zusätzliche Kontrolle durch den Bediener. Im Verlauf der Entwicklung hat sich das Einblenden eines Fadenkreuzes in die Mitte beider Kamerawiedergaben als hilfreich erwiesen. Dies hilft bei manuellen Positionierungen, zum Beispiel an Kanten oder Referenzpunkten.

- Manuelle Steuerung der X/Y/Z/E-Achse:
  Für die manuelle Steuerung bot sich die Verwendung eines Steuerkreuzes für die X- und Y-Achse, sowie von zwei Steuerpfeilen für die Z-Achse an. Eine Steuerung der Rotation der E-Achse mithilfe eines Drehregler wurde wieder verworfen, da sich dies für den Anwendungszweck als nicht zweckmäßig herausstellte. Die Manuelle Steuerung soll in erster Linie dazu dienen den Greifarm grob zu bewegen, zum Beispiel um ihn für ein Eingreifen in die Arbeitsfläche aus dem Weg zu fahren. Die Rotation der E-Achse ist dafür nicht notwendig. Zusätzlich zu dem Steuerkreuz können alle Achsen auch per Eingabe einer Strecke in Millimeter (mit zwei Nachkommastellen) oder eines Winkels in Grad positioniert werden. Dabei kann zwischen relativer oder absolut zu verfahrender Strecke gewählt werden. Dies ermöglicht eine sehr genaue Steuerung, welche vor allem für manuelle Kalibrierungen oder Messungen notwendig ist.

- Möglichkeit zur Kalibrierung:
  Da der mechanischer Aufbau des LitePlacers keine idealen Voraussetzungen im Bezug auf Winkligkeit oder Maßhaltigkeit bietet, ist es nicht möglich, dauerhafte Zielpositionen nur einmal einzumessen und dann unveränderbar zu belassen. Die ursprünglich eingemessenen Koordinaten würden ihre Gültigkeit verlieren, sobald sich die Konfigurationen, zum Zeitpunkt des Einmessens, durch mechanischen Verschleiß oder äußere Einwirkung verändert hätten. Dadurch ist es notwendig, dass wichtige Punkte auch im Nachhinein neu eingestellt werden können. Durch die Kalibrierung werden Referenzmarker automatisch angefahren. Mithilfe der Kamera wird der Mittelpunkt dieser Marker erkannt und dieser als neuer Referenzpunkt abgespeichert. Dies ermöglicht auch ein einfaches Umstrukturieren der Arbeitsfläche, da nur die Koordinaten in einer Konfigurationsdatei auf die Lage der Referenzmarker angepasst werden müssen.

- Laden von Tray abhängigen Parametern:
  Um verschiedenen Arten von ICs testen zu können, müssen die modellspezifischen

Parameter aus Konfigurationsdateien geladen werden. Zur einfacheren Bedienung erfolgt die Wahl der Datei per Dateibrowser.

- Information und Kontrolle des Testablaufs:
  Sie soll dem Benutzer Informationen über den aktuellen Testverlauf geben und den Start und Abbruch einer Testroutine erlauben. Wichtige Kenngröße ist unter anderem die bisherige Durchfallquote.

Im Verlauf der Entwicklung ergab sich die Notwendigkeit für weitere Bedien- und Informationselementen:

- Automatische Anfahrt von vordefinierten Punkten:
  Wichtige Orte, welche wiederholt angefahren werden müssen, sind der Testsockel sowie die Tischkamera.
- Aufheben und Ablegen eines ICs:
  Aufgrund des räumlichen Versatz zwischen dem Saugheber und der Fahrtkamera (siehe Abb. 5.1) ist es nicht möglich, den Greifer per Kamera über einem IC zu positionieren um ihn aufzunehmen. Stattdessen muss zuerst der Versatz der X- und Y-Achse ausgeglichen und dies nach dem Anheben beziehungsweise Ablegen rückgängig gemacht werden, damit die gleiche Position wie vorher wiederhergestellt ist. Zusätzlich muss beim Anheben oder Ablegen zum richtigen Zeitpunkt der Unterdruck ein- oder ausgeschaltet werden. Der Ablauf des Aufnehmens und Ablegens ist in Kapitel 7.4 und 7.5 beschrieben.

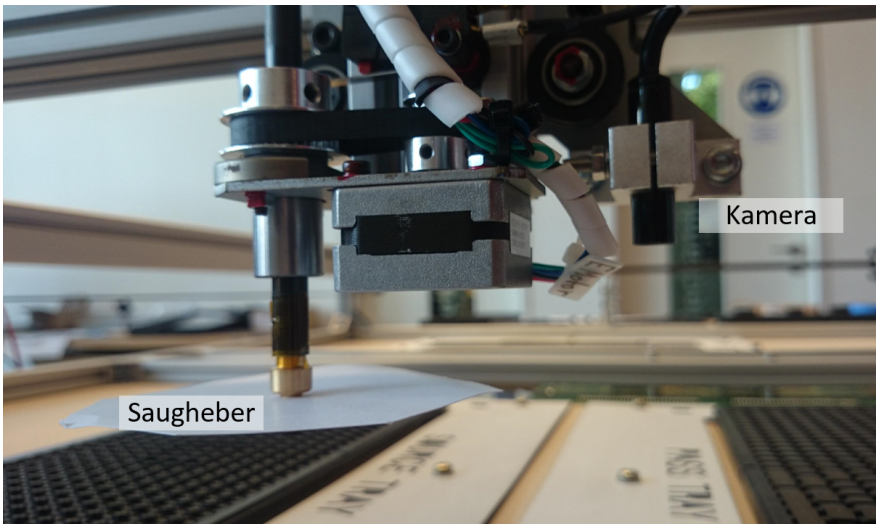


Abbildung 5.1.: Räumlicher Versatz zwischen Kamera und Saugheber (Ansicht von vorne)

- Einstellen der Kameras:
  Da die Kameras mechanisch nicht genau rechtwinklig zu der X- und Y-Achse ausgerichtet werden können (aufgrund von baulich bedingten Eigenschaften), werden mithilfe von Referenzquadraten, die rechtwinklig zu den Achsen liegen, die Kamerabilder ausgerichtet (siehe Abb. 4.7). Erst mit diesem Korrekturwinkel sind die Bilder zur Ausrichtung der ICs geeignet. Zusätzlich können beide Kamerabilder in 90°Schritten gedreht werden, falls dies für den Benutzer einen Vorteil bietet.
- Hardware Einstellungen:
  Zu Beginn der Entwicklung stand nicht fest, welches Steuermodul genutzt werden soll. Deshalb wurde die Möglichkeit eingebaut, aus den angeschlossenen TRINAMIC Motion Controllern den gewünschten auszuwählen und die Antriebe des Testhandlers einer jeweiligen Achse des Moduls zuzuordnen. Dadurch ist es auch möglich, den ursprünglich eingebauten Motion Controller gegen einen anderen auszutauschen, solange dieser genügend Motorausgänge und Steuerausgänge bietet. Außerdem ist es möglich, anstatt einer Steuerelektronik mit mehreren Achsen, mehrere mit jeweils einer Achse (zum Beispiel ein TRINAMIC PANdrive Modul) zu verwenden. In das Programm muss dafür nicht eingegriffen werden.

5.3. Bedienoberfläche der Steuerungssoftware

Die Benutzeroberfläche wurde der Übersichtlichkeit halber in Registerkarten aufgeteilt. Diese haben unterschiedliche Aufgabenbereiche und werden im Folgenden beschrieben.

5.3.1. Registerkarte Control

Die erste Registerkarte enthält alle Anzeige- und Steuerelemente. Von hier wird der Testhandler, nachdem er eingerichtet wurde, gesteuert. Es werden aktuelle Informationen dargestellt, wie zum Beispiel die Livebilder der Kamera oder die aktuellen Positionen der Achsen. Im Folgenden sind die einzelnen Bereiche aufgelistet und beschrieben. Die Bereiche sind in Abbildung 5.2 markiert. Dabei entspricht die Nummer des Bereichs der Nummerierung der Aufzählung.

1. Kamera Livebild und Kamerasteuerung:
   Beinhaltet die Wiedergabe der beiden Kameras, sowie die Möglichkeit, die Kamerabilder um jeweils $\pm 90^{\circ}$zu rotieren. Die Kamerakalibrierung kann genutzt werden, um eine verdreht eingebaute Kamera auszugleichen (siehe Abb. 4.7)

2. Manuelle Steuerung:
   Die manuelle Steuerung besteht aus mehreren Teilen:
   - Einem Steuerkreuz zur intuitiven Bewegung.
   - Einer Homing Taste je X-, Y- und Z-Achse.
   - Einer automatisierten Aufnahme- und Ablageroutine von ICs.
   - Mehrere Funktionen, um einen angehobenen IC manuell auf Ausrichtung und Versatz zu überprüfen.
   - Tasten zur direkten Positionierung über der Tischkamera oder dem Testsockel.
   - Eine Ausgabe der aktuellen Positionen aller vier Achsen in Millimetern mit vier Nachkommastellen.
   - Für jede Achse kann eine zu verfahrende Strecke oder eine anzufahrende Position eingegeben werden.
3. Tray Steuerung:
   Es gibt für alle drei Trays die Optionen zu dem jeweiligen Nullpunkt des Trays oder zu einem über, die Reihe und Spalte im Tray ausgewählten, IC zu fahren.
4. Automatische IC-Testroutine starten und anhalten.
5. Informationen über den aktuellen Testverlauf:
   In diesem Bereich werden die aktuellen Informationen der Testroutine dargestellt, unter anderem, wie viele ICs die Testroutine bestanden haben und wie viele nicht.

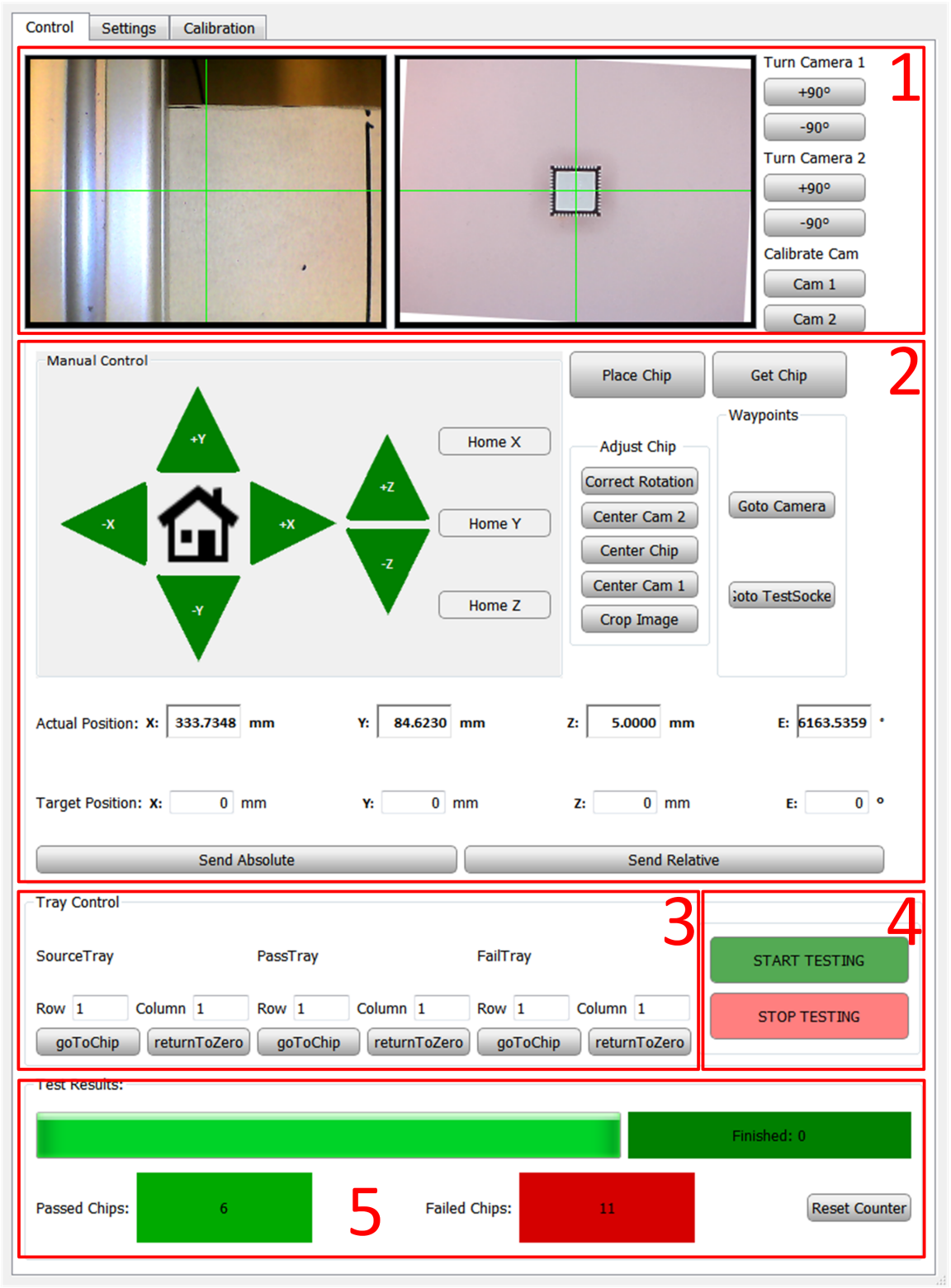


Abbildung 5.2.: Registerkarte Control

5.3.2. Registerkarte Settings

Die zweite Registerkarte beinhaltet die manuellen Hardware Einstellungen und die Möglichkeit, fertige Konfigurationen zu laden.

1. Auswahl der angeschlossenen Steuermodule sowie manuelle Einstellungen und die Möglichkeit eine Hardware Konfigurationsdatei zu laden.
2. Laden der Konfigurationsdatei des zu testenden ICs und der dazugehörenden Mustervorlage.
3. Abbildung der geladenen Mustervorlage.

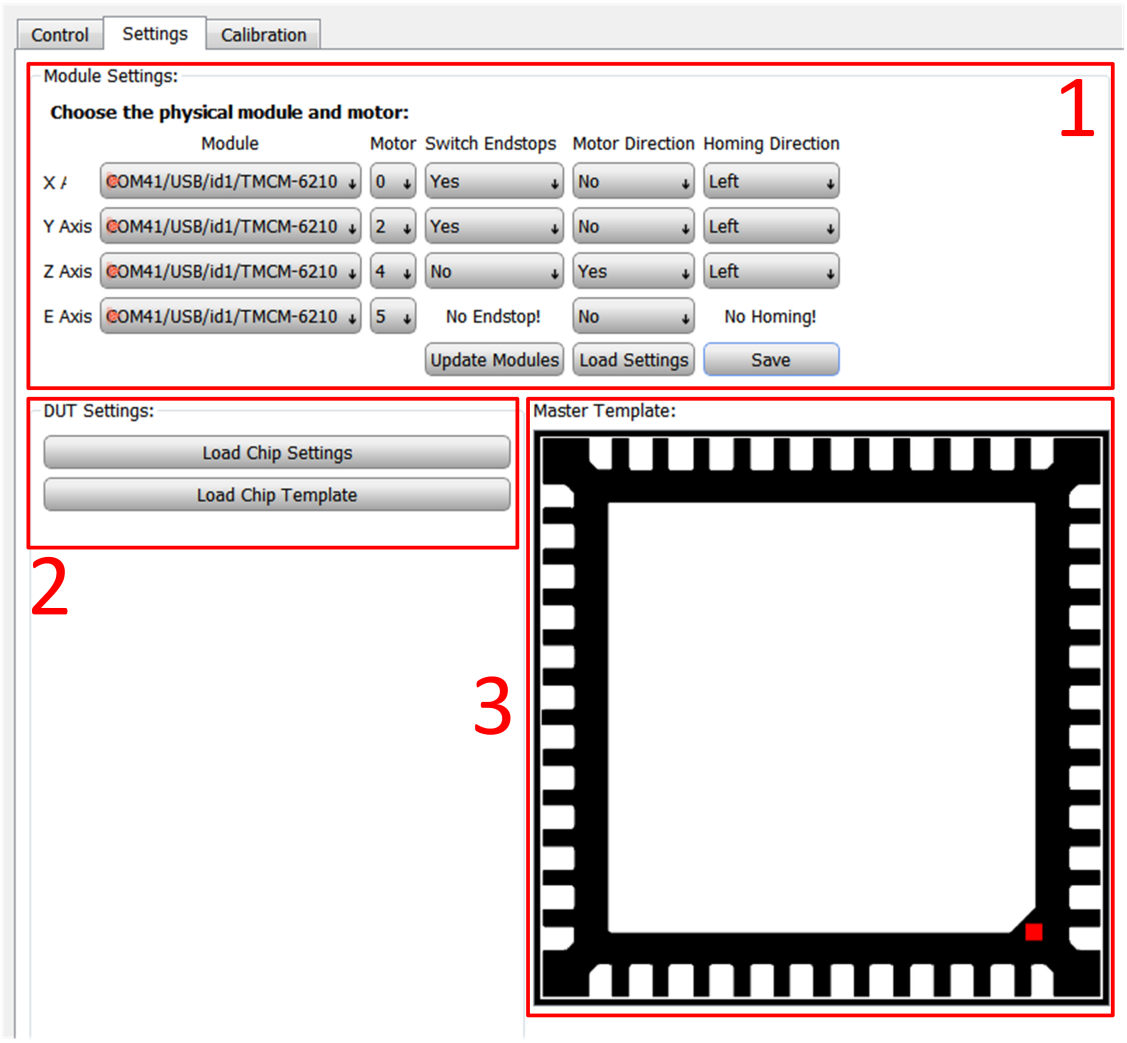


Abbildung 5.3.: Registerkarte Settings

5.3.3. Registerkarte Calibration

In der dritten Registerkarte können verschiedene Kalibrierungen gestartet werden. Das ist jedoch nur für eine manuelle Bedienung notwendig, da die IC-Testroutine eine automatische Kalibrierung beinhaltet.

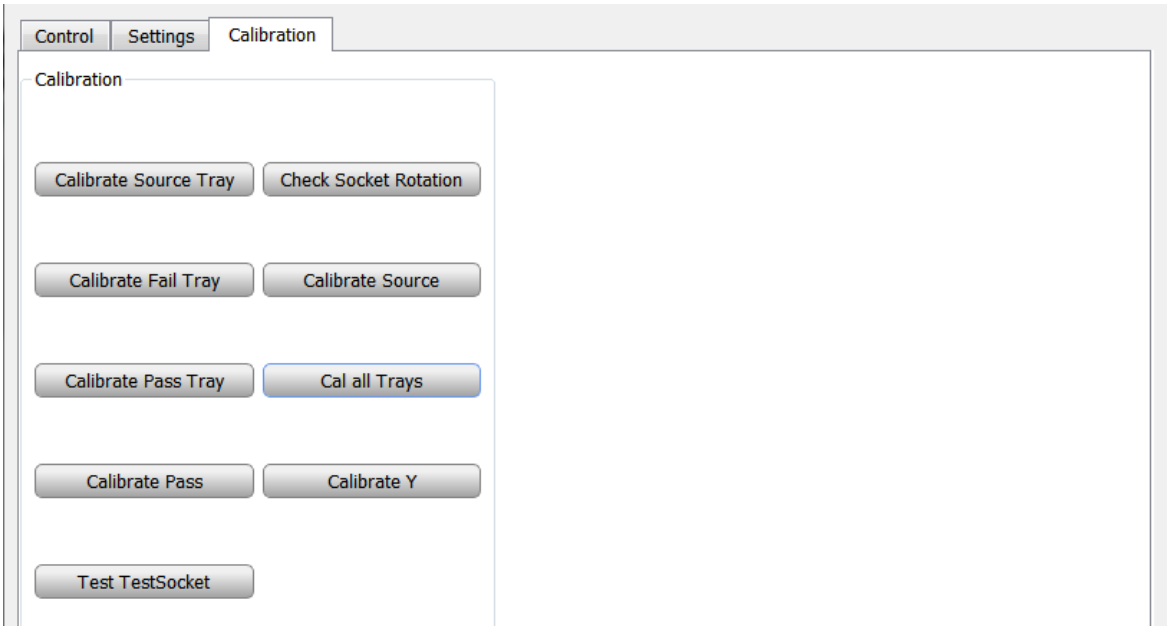


Abbildung 5.4.: Registerkarte Calibration

6. Aufbau des Testhandler Plug-in

Die Software des Testhandlers ist als Plug-in der TMCL-IDE von TRINAMIC entwickelt worden, wodurch eine modulare Entwicklung und Wartung möglich ist.

Auf die einzelnen entwickelten Klassen und Bibliotheken wird im Folgenden eingegangen.

6.1. Testhandler Klasse

Die *Testhandler* Klasse basiert auf der *ContainerInterface* Klasse von TRINAMIC und kann dadurch als Plug-in in die TMCL-IDE eingebunden werden. Sie ist unter Listing E.5 und E.6 abgebildet. Diese Klasse verbindet die entwickelten Klassen und Bibliotheken zusammen mit der Benutzeroberfläche und den verschiedenen Zustandsautomaten. Die *Testhandler* Klasse besteht aus den mehreren Elementen, welche der Übersichtlichkeit halber auf verschiedenen Dateien, Klassen und Bibliotheken aufgeteilt wurden. Folgende Elemente sind in der *Testhandler* Klasse enthalten:

- Graphische Benutzeroberfläche:
  Ermöglicht eine einfache Bedienung des Testhandlers und gibt dem Benutzer Informationen über den aktuellen Stand. Siehe Listing E.8.
- Timer Event:
  Der Timer Event wird alle 30 ms ausgelöst. Dabei werden neue Bilder von den beiden Kameras geladen und in der Benutzeroberfläche dargestellt. Dadurch kommen die Livebilder zustande. Siehe Listing E.11.
- Zustandsautomaten:
  Die *Testhandler* Klasse enthält mehrere Zustandsautomaten die genutzt werden können. Beispiele dafür sind die IC-Testroutine oder die automatische Referenzierung der Trays. Siehe Listing E.9.
- Konfiguration und Kalibrierung:
  Zum Einrichten des Testhandlers existieren Funktionen um Konfigurationsdateien zu laden und auszuwerten. Weiterhin gibt es Funktionen zur Kalibrierung des Testhandlers. Siehe Listing E.10 und E.7.

- *AxisConfiguration*:
  Für die vier Achsen des LitePlacers werden vier Objekte der *AxisConfiguration* Klasse erzeugt (siehe Kapitel 6.2 und Listing E.3 und E.4).
- *Tray*:
  Für die drei Trays werden drei Objekte der *Tray* Klasse erzeugt (siehe Kapitel 6.3 und Listing E.12 und E.13).
- *Testsocket*:
  Um eine elektronischen IC-Test zu emulieren wird ein Objekt der *Testsocket* Klasse erzeugt (siehe Kapitel 6.4 und Listing E.14 und E.15).
- *ImageProcessing*:
  Die Funktionen der entwickelten *ImageProcessing* Bibliothek werden für die Bildverarbeitung und Erkennung verwendet. Siehe Listing E.1 und E.2.

Eine graphische Übersicht der *Testhandler* Klasse wird in Abbildung 6.1 dargestellt. Die Namen an den Pfeilen sind die erzeugten Objekte der jeweiligen Klasse.

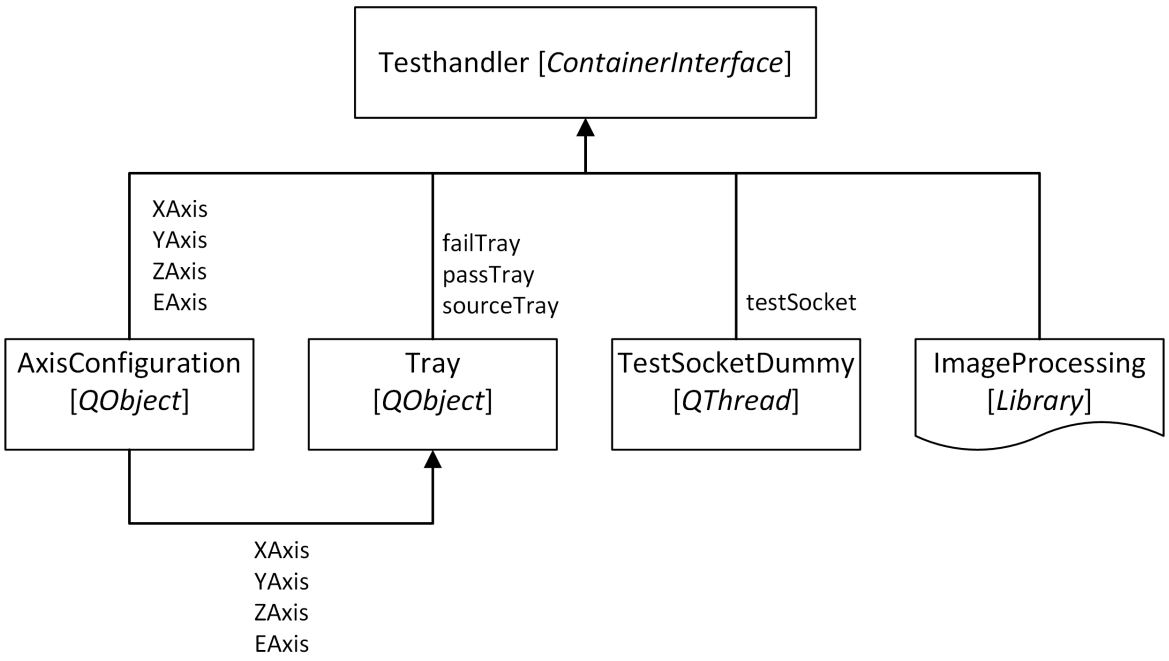


Abbildung 6.1.: Übersicht über das Testhandler Plug-in

6.2. AxisConfiguration Klasse

Die *AxisConfiguration* Klasse basiert auf der *QObject* Klasse von Qt und repräsentiert eine Motorachse. Durch die Verwendung dieser Klasse für jede Achse, ist eine einfache Anpassung der Funktionen möglich, ohne jede Achse einzeln ändern zu müssen. Einem Objekt dieser Klasse kann ein Steuermodul und, falls ein Module mit mehreren Achsen verwendet wird, der zu verwendende Motorausgang zugeteilt werden. Die Kommunikation mit der Steuerelektronik von TRINAMIC erfolgt mithilfe von TMCL-Befehlen, die in den entsprechenden Funktionen parametrisiert und gesendet werden. Für einen Übersicht der TMCL-Befehle und ihrer Funktionen sei auf den Anhang A.2.2 und A.2.3 sowie auf [16] verwiesen. Die Klasse enthält verschiedene Funktionen und Parameter, die im Folgenden beschrieben werden:

- Bewegungsparameter:
  Die Motion Controller Elektronik benötigt mehrere Geschwindigkeits- und Beschleunigungsparameter pro Motor um ihre internen Berechnungen durchzuführen. Diese unterscheiden sich je nach Achse, was mechanische Gründe hat. So muss die E-Achse, die den Saugheber rotiert, eine andere maximale Geschwindigkeit haben als die X-Achse.Diese Parameter sind in der Hardware Konfigurationsdatei für jede Achse gespeichert und können angepasst werden.

- Mechanische Parameter:
  Die verschiedenen Achsen des LitePlacers unterscheiden sich in ihren Vortriebsarten. Die X- und Y-Achse werden über Zahnriemen angetrieben, die Z-Achse über eine Gewindespindel und die E-Achse über einen Zahnriemen mit Untersetzung. Diese Parameter sind in der *AxisConfiguration* Klasse vorhanden und einstellbar. Zusätzlich ist auswählbar, welche Endschalter verwendet werden und ob die Angabe der zu verfahrenden Strecke in Millimetern oder in Grad (für die E-Achse) erfolgt. Diese Parameter sind in der Hardware Konfigurationsdatei für jede Achse gespeichert und können angepasst werden.

- Bewegungsfunktionen:
  Um die Ansteuerung der Achsen zu vereinfachen wurde folgende Funktionen in die *AxisConfiguration* Klasse integriert:
  - Verfahren der Achse durch die Angabe einer Zielposition in Millimetern oder Grad. Dadurch werden beim Funktionsaufruf die Bahnberechnung innerhalb der Klasse durchgeführt. Die entsprechende Funktion ist `moveToPosition`.
  - Verfahren der Achse durch die Angabe einer Geschwindigkeit und Richtung. Die entsprechende Funktion ist `velocityMode`.
  - Homing der Achse. Die entsprechende Funktion ist `home`.
  - Anhalten der Achse. Die entsprechende Funktion ist `motorStop`.
- Informationsfunktionen:
  Diese geben Parameter der Achse, wie zum Beispiel die aktuelle Geschwindigkeit, zurück . Eine beispielhafte Funktion ist `actualPosition`.
- I/O Funktionen:
  Diese Funktion kommuniziert mit den steuerbaren Ein- und Ausgängen des Moduls. Die entsprechende Funktion ist `setIO`.

Von dieser Klasse werden vier Objekte erzeugt, eins für jede Achse.

6.3. Tray Klasse

Durch die Verwendung von drei gleichen Trays bot sich die Entwicklung einer entsprechenden Klasse an, da alle drei Trays die gleichen Funktionen benötigen. Die *Tray* Klasse basiert auf der *QObjects* Klasse von Qt. Die in der *Tray* Klasse enthaltenen Funktionen und Parameter sind im Folgenden beschrieben:

- IC-spezifische Parameter:
  Je nach Gehäuse des ICs unterscheiden sich die Anzahl an Reihen und Spalten eines Trays. Dadurch ändert sich auch der Abstand der ICs in X- und Y-Richtung. Diese Werte sowie der Name des verwendeten ICs werden der Klasse beim Laden aus der IC spezifischen Konfigurationsdatei übergeben. Eine beispielhafte Funktion ist `setRows` .
- Funktionen der Bahnsteuerung:
  Die *Tray* Klasse besitzt eigene Funktionen der Bahnsteuerung. Diese ermöglichen es über Funktionen wie *goToNextChip()* zeilenweise den nächsten IC anzufahren, oder über *returnToZero()* zu der Nullposition der Trays zurückzukehren. Dadurch entfallen

Berechnungen der genauen Koordinaten außerhalb der Klasse, was eine einfachere Nutzung ermöglicht.

- Ausrichtung:
  Für den Aufbau des Testhandler wurden das Source- und das Passtray horizontal, das Failtray jedoch vertikal befestigt. In der *Tray* Klasse kann ausgewählt werden, welche Orientierung das Tray hat. Die interne Koordinatenberechnung passt sich an die gewählte Ausrichtung an, wodurch es für die Verwendung eines *Tray* Objektes keinen Unterschied macht, wie das Tray orientiert ist. Die entsprechende Funktion ist `setVertical`.
- Einen Zähler der enthaltenen ICs:
  Jedes mal wenn ein IC aus dem Tray entfernt oder hinzugefügt wird, wird ein Zähler dekrementiert beziehungsweise inkrementiert. Dadurch kann erkannt werden, wann ein Tray voll oder leer ist.

6.4. Testsocket Klasse

Die *Testsocket* Klasse ist eine Emulation der Testelektronik. Da es zum Zeitpunkt der Entwicklung des Testhandlers nicht möglich war, eine echte Testelektronik zu verwenden und einzubinden, wurde eine Klasse entwickelt die diese emuliert. Ein Objekt diese Klasse kann wie die echte Testelektronik gestartet werden und gibt nach einer kurzen Zeit ein Ergebniscode zurück. In dieser Emulation wurde eine Durchfallquote von 20% ausgewählt, um eine Verteilung in Fail- und Passtray zu ermöglichen.

6.5. ImageProcessing Softwarebibliothek

Diese Bibliothek wurde entwickelt, um die benötigten Funktionen der Bildverarbeitung und Erkennung geordnet zu verwalten. Die beinhalteten Funktionen werden großteils in Kapitel 4 beschrieben. Im Folgenden ist eine Übersicht über die wichtigsten, enthaltenen Funktionen gegeben:

- *getDegree*:
  Erkennt den Winkel, um den das größte Rechteck auf einem Bild rotiert ist.
- *getDeltaX* und *getDeltaY*:
  Gibt in Pixeln die Differenz zwischen dem Mittelpunkt des Bildes und dem Mittelpunkt des größten Rechteckes in dem Bild in X- beziehungsweise Y-Richtung an.

- *getBorders*:
  Erkennt die Kanten des größten Rechteckes auf einem Bild und gibt diese als `RotatedRect` zurück. Eine genaue Beschreibung ist in Kapitel 4.3 nachzulesen.
- *cropImg*:
  Schneidet aus einem Bild eine angegebene Fläche aus und gibt diese als `Mat` Datentyp zurück.
- *rotateImg*:
  Rotiert ein übergebenes Bild um einen übergebenen Winkel in Grad.
- *compareImage*:
  Vergleicht zwei Bilder an einer ausgewählten Fläche miteinander. Es bereitet die Bilder für die *compareMat* Funktion vor und ruft diese auf. Die Funktion gibt einen normierten Wert zwischen 0 und 1 zurück, wobei eine 0 bedeutet, dass die zu vergleichende Fläche absolut identisch sind. Eine 1 bedeutet , dass keine Pixel der Fläche miteinander übereinstimmten. Die genaue Funktion ist in Kapitel 4.6.3 beschrieben.
- *compareMat*:
  Diese Funktion vergleicht zwei binäre, gleichgroße Matrizen pixelweise miteinander. Die genaue Funktion ist in Kapitel 4.6.3 beschrieben.
- *generateEvenBackround*:
  Diese Funktion schneidet eine angegebene Fläche aus einem Bild aus und erzeugt basierend auf der Farbe am äußeren Rand des Ausschnittes einen gleichmäßigen Hintergrund für diesen Ausschnitt. Die genaue Funktion ist in Kapitel 4.7 beschrieben.
- *extractColor*:
  Erkennt in einem übergebenen Bild eine gesuchte Farbe. Es wird ein Bild zurückgegeben, in dem nur die Pixel vorkommen, der gesuchten Farbe entsprechen.
- *pixelToMillimeter*:
  Berechnet aus einem Bild von einem Quadrat mit bekannter Kantenlänge einen Faktor. Dieser wird genutzt, um berechnen zu können, wie viel Millimeter ein in Pixeln gemessenes Objekt groß ist. Die genaue Funktion ist in Kapitel 4.5.2 beschrieben.

7. Konfiguration und Ablaufsteuerung

Der Testhandler muss vor seiner Benutzung zuerst Konfiguriert und Kalibriert werden, damit die Bahnberechnungen stimmen. Dafür können Konfigurationsdateien geladen sowie automatische Kalibrierungen durchgeführt werden.

7.1. Automatische Kalibrierung und Referenzierung

Bevor die Testroutine gestartet werden kann, muss der Software bekannt sein, wo sich die Achsen aktuell befinden und welche Koordinaten die Zielpositionen haben. Zusätzlich muss der nicht rechtwinklige Aufbau des LitePlacers kompensiert werden können. Zu diesem Zweck werden Kalibrierungsfahrten absolviert, welche in den nachfolgenden Kapiteln beschrieben werden.

7.1.1. Homing

Das sogenannte Homing beschreibt das Anfahren und Auswerten der Endschalter einer Achse und das darauffolgende Anfahren dessen Nullpunktes. Durch das Auswerten der Endschalter ist der Software bekannt, wie groß die maximal verfahrbare Strecke der Achse ist und wo sich Start- und Endpunkt befinden. Diese werden benötigt, um Bahnberechnungen durchführen zu können.

7.1.2. Bestimmung des Kompensationsfaktors der Y-Achse

Um die in Kapitel 4.5.1 beschriebene Methode zur Kompensation von Winkelungenauigkeiten des LitePlacers durchzuführen, müssen zwei Referenzmarker angefahren und deren Koordinaten bestimmt werden. Genutzt werden die beiden Referenzmarker des Failtray. Die Koordinaten der Marker sind in der Hardware-Konfigurationsdatei hinterlegt und können an eine veränderte Position des Trays angepasst werden. Dabei müssen die vorgegebenen Koordinaten nicht exakt mittig auf den Markierungen liegen. Es reicht, wenn die Fahrtkamera

die Marker vollständig im Bild hat, wodurch eine automatische Zentrierung möglich ist. Diese Routine wird vor der IC-Testroutine automatisch durchgeführt. Der Ablauf dieser Routine ist in Abb. 7.1 dargestellt. Die entsprechende Funktion heißt `measureYCompansation` (siehe Listing E.10).

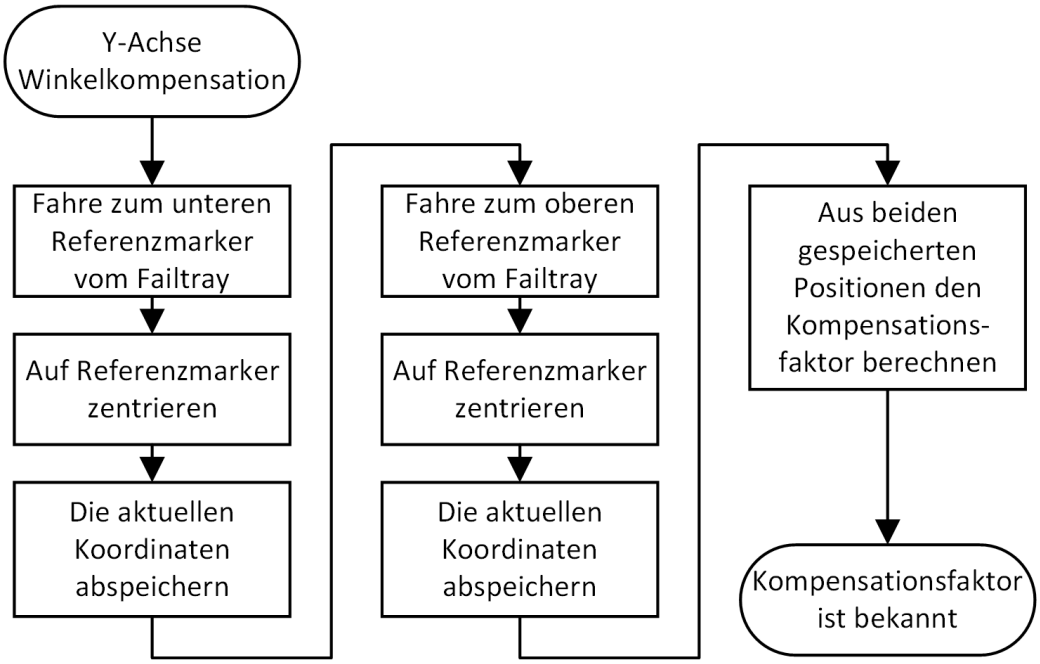


Abbildung 7.1.: Automatische Bestimmung des Kompensationsfaktors der Y-Achse

7.1.3. Einmessen der IC-Trays

Um die Trays wie in Kapitel 4.5 beschrieben automatisch Einmessen zu können, gibt es einen automatischen Kalibrierungsvorgang. Dabei werden die Referenzmarker der drei Trays angefahren und dessen Positionen erfasst. Wie in Kapitel 7.1.2 beschrieben, sind die Koordinaten der Referenzmarker für das Fail-, Source- und Passtray in der Hardware-Konfigurationsdatei abgespeichert. Diese Routine wird vor der IC-Testroutine automatisch durchgeführt. Der Ablauf ist in Abb. 7.2 dargestellt. Die entsprechende Funktionist `calAllTrays` in Listing E.9.

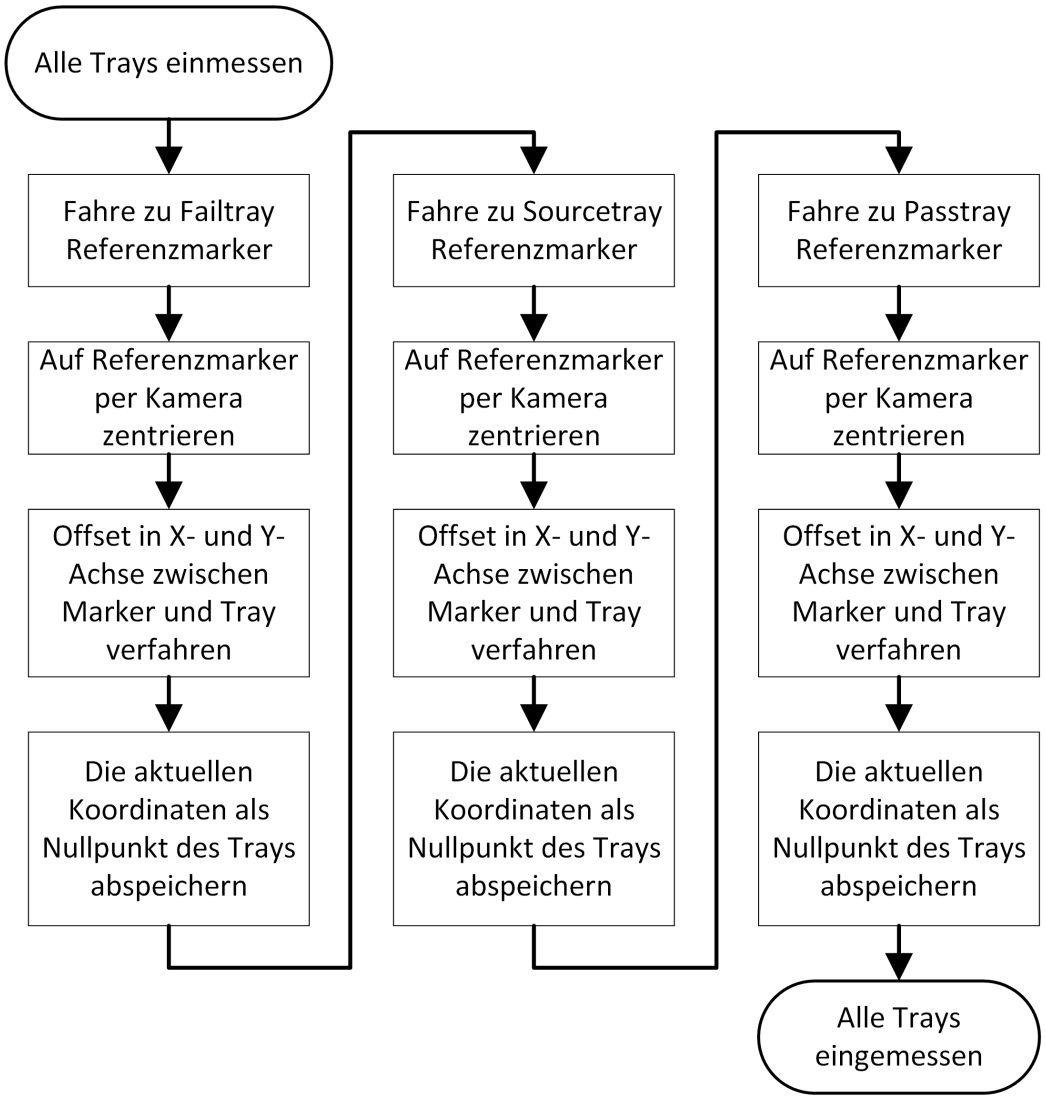


Abbildung 7.2.: Automatische Kalibrierung der drei Trays

7.2. Konfigurationsdateien

Eine Konfigurations- oder Initialisierungsdatei ist eine Textdatei mit der Dateiendung ".ini". Sie enthält Wertepaare, welche zusätzlich auch in Gruppen eingeteilt werden können. Die Zeichenfolge links von dem Gleichheitszeichen bilden den sogenannten Schlüssel und die Zeichenfolge rechts den dazugehörigen Wert. Das Programm welches diese Initialisierungsdatei lädt durchsucht diese nach bestimmten Schlüsselwörtern, die in dem Programm festgelegt wurden. Findet es dieses Wort, speichert es den dazugehörigen Wert in z.B. einer

Variable ab [21]. Der Vorteil ist, dass Werte beziehungsweise Parameter des Programms geändert werden können, ohne den Quellcode zu ändern oder neu zu kompilieren. Zusätzlich bieten Initialisierungsdateien die Möglichkeit viele Werte auf einmal einzulesen. Auch können verschiedene Initialisierungsdateien für dasselbe Programm verwendet werden, die zum Beispiel für verschiedene Anwendungszwecke gedacht sind. Durch die Nutzung von eckigen Klammern können Gruppen erzeugt werden. Es darf jede Gruppe nur einmal geben, allerdings dürfen in verschiedenen Gruppen die gleichen Schlüssel vorkommen. Im folgenden in ein Ausschnitt aus der *hardware.ini* Datei des Testhandlers abgebildet. Zu sehen ist hier die Gruppe `[XAXIS]` und die dazugehörigen Schlüssel und deren Werte. Falls in diesem Fall die Motordrehrichtung geändert werden soll, muss bei `SwitchMotorDirection` das `false` durch ein `true` ersetzt werden.

Listing 7.1: Beispiel einer Konfigurationsdatei

```
1 [XAXIS]
2 Motor=0
3 SwitchEndstops=true
4 SwitchMotorDirection=false
5 PulleyToothCount=20
```

7.2.1. Hardware Konfiguration

Die Hardware-Initialisierungsdatei enthält bestimmte Parameter und Koordinaten, die unabhängig von dem zu testenden IC sind und sich hauptsächlich auf den elektrischen und mechanischen Aufbau des Testhandlers beziehen:

- Die Beschleunigungs-, Geschwindigkeits- und Untersetzungs-Parameter, sowie die Einstellung der Endschalter und Rotationsrichtung für die X/Y/Z/E-Achse.
- Die Koordinaten für die Referenzierungspunkte.
- Die Korrekturfaktoren um mechanische Ungenauigkeiten auszugleichen.
- Die Ausrichtungswinkel der jeweiligen Trays.

Die Initialisierungsdatei für den Testhandler, *hardware.ini*, befindet sich im Anhang unter Listing D.1.

7.2.2. Chip Konfiguration

Um ein bestimmtes IC-Modell testen zu können, sind zwei zusammengehörende Dateien notwendig:

- Eine IC-spezifische Initialisierungsdatei:
  Je nach Art des IC-Gehäuse variieren die Trays in ihren internen Maßen. Das heißt, dass der Abstand von der Tray Außenkante zum ersten IC und die Abstände zwischen den ICs unterschiedlich sind. Dazu kommen noch die Anzahl an vorhandenen Reihen und Spalten. Diese Informationen braucht das Programm, um die richtigen Koordinaten für jeden IC berechnen zu können. Die Initialisierungsdatei enthält deshalb folgende Parameter:
  - Den Namen des IC-Modells.
  - Den Abstand von der Außenkante des Trays zum ersten IC in X- und Y-Richtung.
  - Den Abstand zwischen den ICs in X- und Y-Richtung.
  - Die Anzahl von Reihen und Spalten.

  Eine beispielhafte Initialisierungsdatei für das IC-Modell *TMC5041* befindet sich im Anhang unter D.2.
- Ein Muster Bild von der Unterseite des ICs:
  Das Muster Bild muss von dem Benutzer für das gewünschte Gehäuse angefertigt werden. In Abbildung 7.3 ist eine Mustervorlage für den TMC5041 IC dargestellt. Folgende Vorgaben gelten für die Erstellung einer Mustervorlage:
  - Die Flächen, die bei dem IC aus Metall bestehen (zum Beispiel Anschlussflächen oder Kühlflächen), müssen weiß sein.
  - Die restliche Fläche muss schwarz sein.
  - Die Fläche, die das Merkmal beinhaltet welche die Ausrichtung des ICs angibt (zum Beispiel eine abgeschrägte Ecke der Kühlfläche), muss rot sein.
  - Die Mustervorlage muss 1000 x 1000 Pixel groß sein.

  Die Vorlage ist notwendig, um die Orientierung des ICs zu überprüfen und ihn in die korrekte Ausrichtung zu drehen.

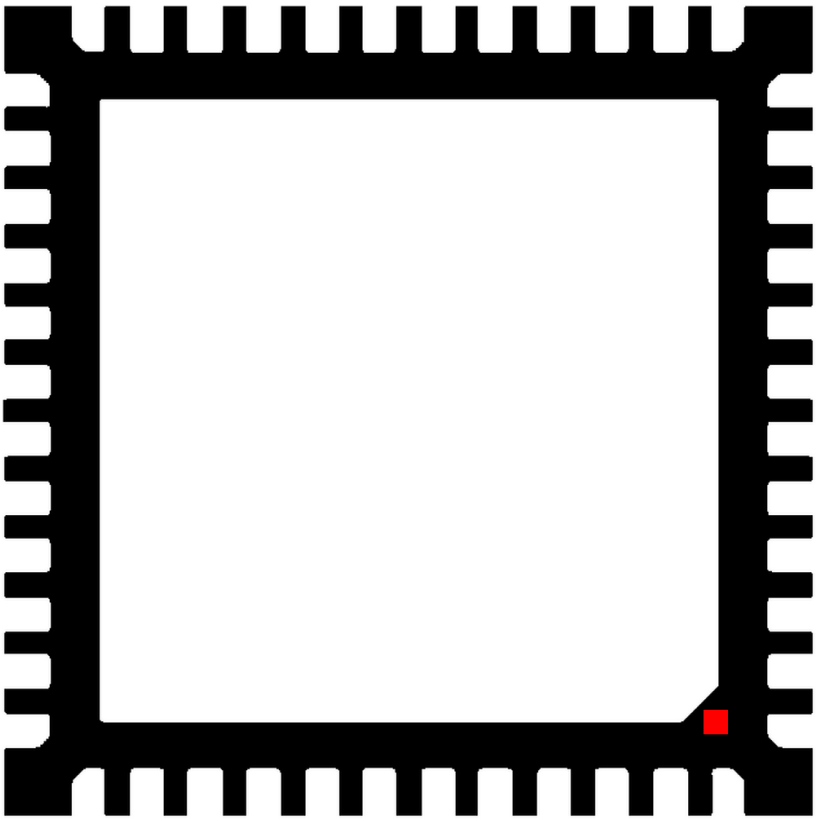


Abbildung 7.3.: Mustervorlage für das IC-Modell *TMC5041* von TRINAMIC

7.3. IC-Testroutine

Die Kernaufgabe des Testhandlers ist es, ICs aus einem Vorratsbehälter (Sourcetray) zu nehmen, sie richtig ausgerichtet und präzise in einer Testvorrichtung zu platzieren und sie nach dem Test, je nach Ergebnis, in ein Tray für funktionsfähige (Passtray) oder defekte (Failtray) ICs zu sortieren. Dabei soll ein komplettes Tray verarbeitet werden können, ohne dass ein menschliches Eingreifen notwendig ist. Die Voraussetzungen für diese Routine sind nachfolgend aufgelistet:

- Die IC-spezifischen Konfigurationsdatei und die richtigen Mustervorlage muss geladen worden sein (siehe Kapitel 7.2.2).
- In die Halterung des Sourcetrays muss ein volles Trays eingelegt worden sein.
- In die Halterungen des Pass- und des Failtrays müssen leere Trays eingelegt worden sein.

- Der Rotationswinkels der eingebauten Kameras (siehe Kapitel 4.4) muss kompensiert worden sein.

Sind diese Voraussetzungen erfüllt, kann die Testroutine gestartet werden. Der Ablauf ist in Abbildung 7.5 dargestellt und wird im Folgenden beschrieben:

1. Es erfolgt die Bestimmung des Kompensationsfaktor, um die Winkelungenauigkeit des mechanische Aufbaus zu kompensieren (siehe Kapitel 4.5.1), und das Einmessen der Trays (siehe Kapitel 7.1.3).

2. Der Testhandler fährt zum Nullpunkt des Sourcetrays und nimmt den aktuellen IC auf, beginnend bei dem ersten IC des Trays (siehe Punkt 1 in Abb. 7.4). Die ICs werden zeilenweise abgearbeitet. Dabei wird jedes mal ein Zähler des Sourcetray Objekts (siehe Kapitel 6.3) um 1 erhöht. Wenn der der Wert des Zählers die Anzahl der ICs pro Tray überschreitet, hat der Testhandler alle ICs des Trays verarbeitet und die Testroutine wird beendet.

3. Der IC wird angehoben und über die Tischkamera gefahren (siehe Punkt 2 in Abb. 7.4).

4. Es wird versucht die Position und Orientierung des ICs zu erkennen (siehe Kapitel 4.6).

5. Falls der IC nach mehreren Versuchen nicht erkannt werden konnte, erfolgt eine Überprüfung, ob der Saugheber überhaupt einen IC angehoben hat. Ist das nicht der Fall, der Saugheber also leer, fährt der Testhandler in die untere linke Ecke der Arbeitsfläche und deaktiviert den Unterdruck (siehe Punkt 3 in Abb. 7.4). Das wird für den Fall gemacht, bei dem ein IC zwar am Saugeheber angesaugt ist, aber die Bilderkennung keinen erkennt. Würde der Unterdruck direkt über der Tischkamera deaktiviert werden, würde der unerkannte IC in den Kameraschacht fallen und könnte die Kamera blockieren. Danach fängt die Routine wieder bei Punkt 2 an. Wenn der Saugheber zwar einen IC angehoben hat, dieser aber nicht erkannt wird, wird dieser in das Failtray sortiert (siehe ab Punkt 7 dieser Beschreibung und Punkt 4 in Abb. 7.4).

6. Wurde der IC richtig erkannt, wird er zum Testsockel gefahren, der gemessene Versatz in X- und Y-Richtung ausgeglichen (siehe Kapitel 4.6.1), darin platziert und der Test gestartet (siehe Punkt 5 in Abb. 7.4).

7. Je nach Ergebnis des Tests wird der IC entweder zu dem Nullpunkt des Fail- oder Passtrays gefahren (siehe Punkt 6a und 6b in Abb. 7.4).

8. Hier erfolgt wieder eine Überprüfung des internen Zählerstandes des Fail- oder Passtrays. Falls das Tray voll sein sollte, wird die Testroutine beendet. Ansonsten fährt der Testhandler zeilenweise zu dem nächsten, freien Feld, um die Trays lückenlos aufzufüllen (siehe Punkt 7a und 7b in Abb. 7.4).

9. Der IC wird so rotiert, dass die Ausrichtung des ICs mit der Ausrichtung des Trays übereinstimmt.
10. Der gemessen Versatz zwischen Saugheber und Mittelpunkt des ICs wird ausgeglichen (siehe Kapitel 4.6.1) und der IC im Tray abgelegt.
11. Danach springt die Routine wieder zu Punkt 2 und beginnt erneut.

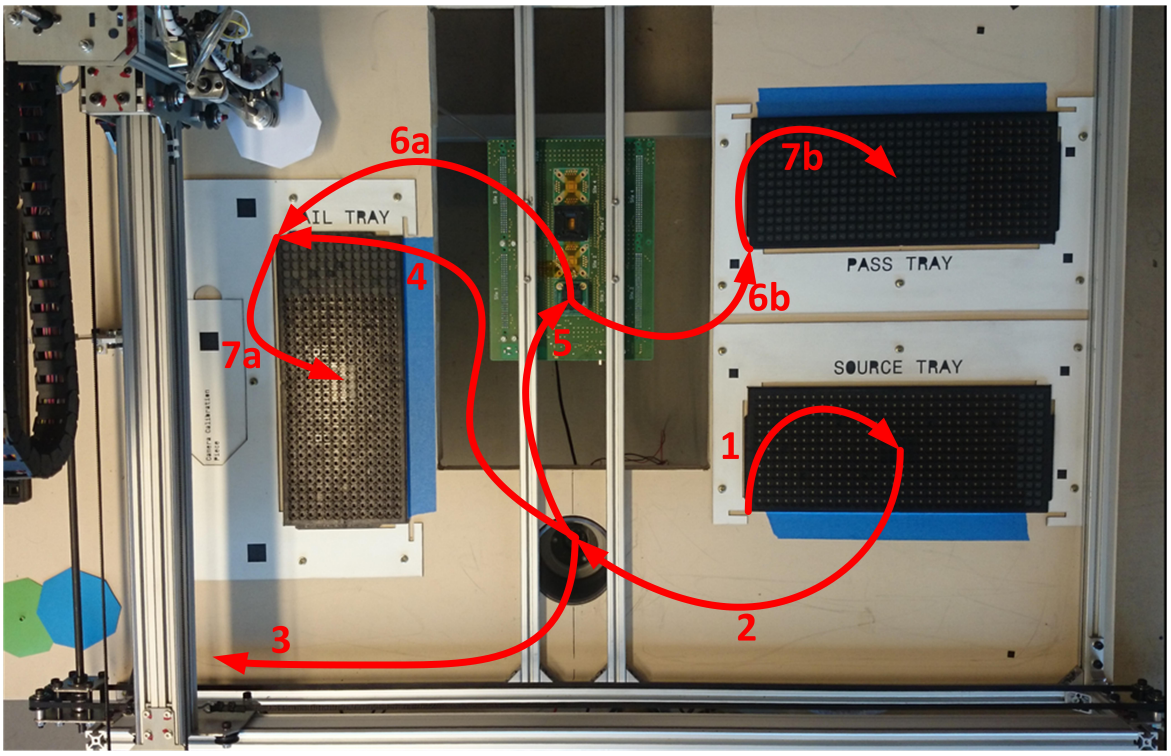


Abbildung 7.4.: Räumliche Ablaufübersicht der Testroutine

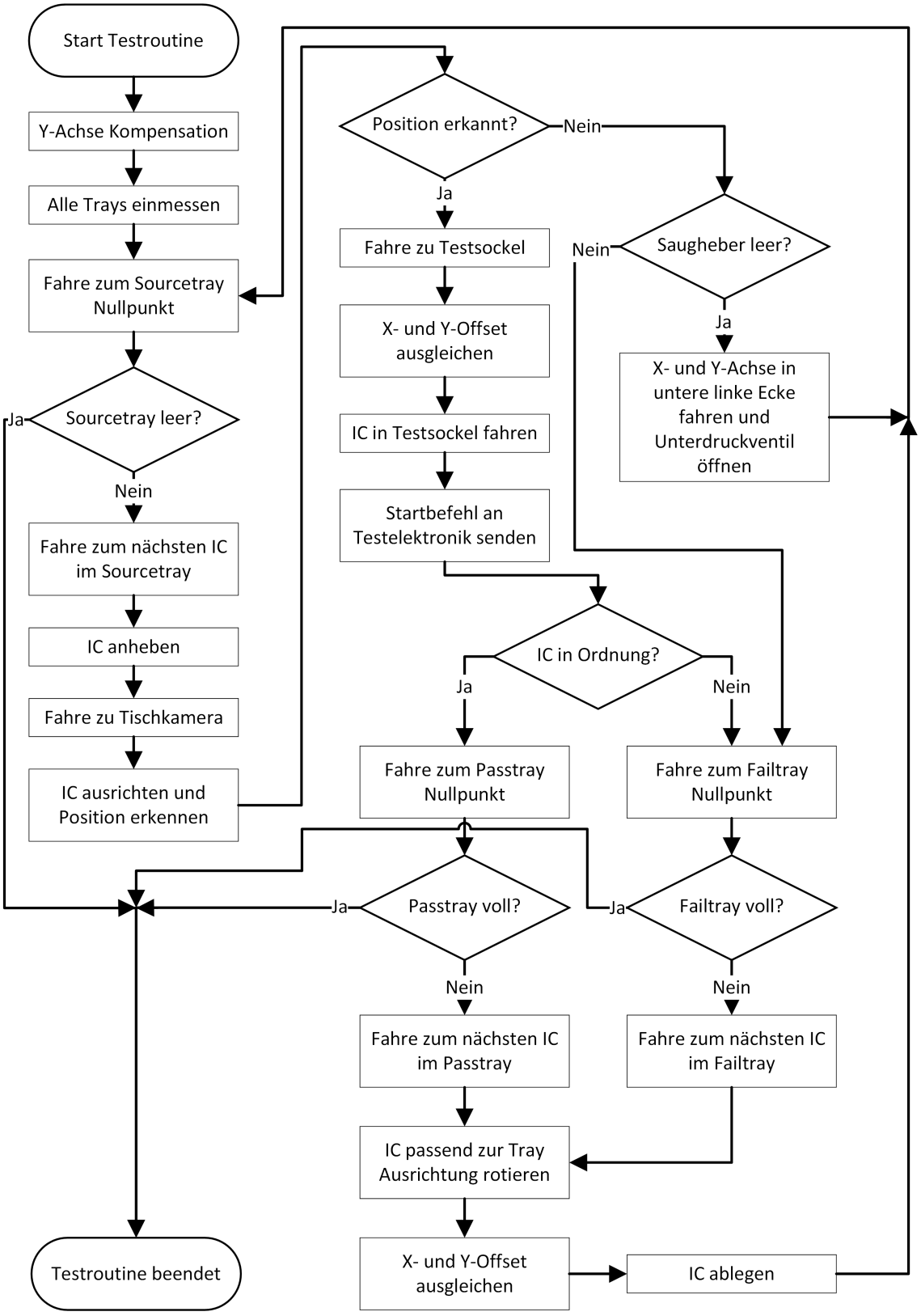


Abbildung 7.5.: Ablaufdiagramm der IC-Testroutine

7.4. Aufnehmen des ICs

Um einen IC per Saugheber anheben zu können, muss dieser mittig und mit eingeschalteten Unterdruck auf den IC abgesenkt werden. Durch einen Endschalter am Saugheber wird erkannt, ob dieser auf etwas aufgefahren ist. Wird dies registriert, wird der Saugheber wieder hochgefahren. Aufgrund des mechanischen Aufbaus des LitePlacers befindet sich zwischen der Fahrtkamera und dem Saugheber jeweils ein Versatz in X- und Y-Richtung. Das führt dazu, dass ein Objekte, auf das die Fahrkamera zentriert ist, nicht direkt mit dem Saugheber angehoben werden können. Zuvor muss der Versatz ausgeglichen werden. Dieser Versatz ist eine konstante Größe, er verändert sich nicht im Laufe des Testablaufs und ist konstruktionsbedingt vorgegeben und kann nicht verändert werden. Er beträgt in X-Richtung -32.1 mm und in Y-Richtung 83.5 mm. Um die ehemalige Position des Testhandler wiederherzustellen wird nach dem Aufnehmen des ICs der Versatz negativ verfahren. Dadurch befindet sich die Fahrtkamera wieder mittig über der ehemaligen Position des aufgenommenen ICs. Ein Nachteil dieses mechanischen Aufbaus ist, dass mit Hilfe Der Fahrtkamera nicht erkannt werden kann, ob der IC tatsächlich angehoben wurde. In der Software kann das Anheben eines ICs durch zwei Funktionen erfolgen. Entweder durch die `on_getICpushButton_clicked` Funktion, zum manuellen Anheben einse ICs (siehe Listing E.8), oder durch die Funktion `getChip` für das Anheben der ICs während der Testroutine (siehe Listing E.13). Der komplette Ablauf ist in Abb. 7.6 dargestellt und wird im Folgendem beschrieben:

1. Die Voraussetzung ist, dass die Fahrkamera sich mittig über dem aufzuhebenden Objekt befindet.
2. Der Versatz zwischen dem Saugheber und der Kamera wird ausgeglichen und die Z-Achse wird langsam nach unten gefahren.
3. Durch das Auffahren des Saughebers auf das anzuhebende Objekt wird dessen Endschalter ausgelöst und die Z-Achse stoppt.
4. Der Unterdruck wird aktiviert und es werden 200 ms gewartet, bis der IC angesaugt wurde.
5. Die Z-Achse wird wieder hoch, bis auf 5 mm an den oberen Endschalter. Das verhindert ein Verklemmen des Endschalters.
6. Die ehemalige Position wird wiederhergestellt, indem der Versatz negativ verfahren wird.

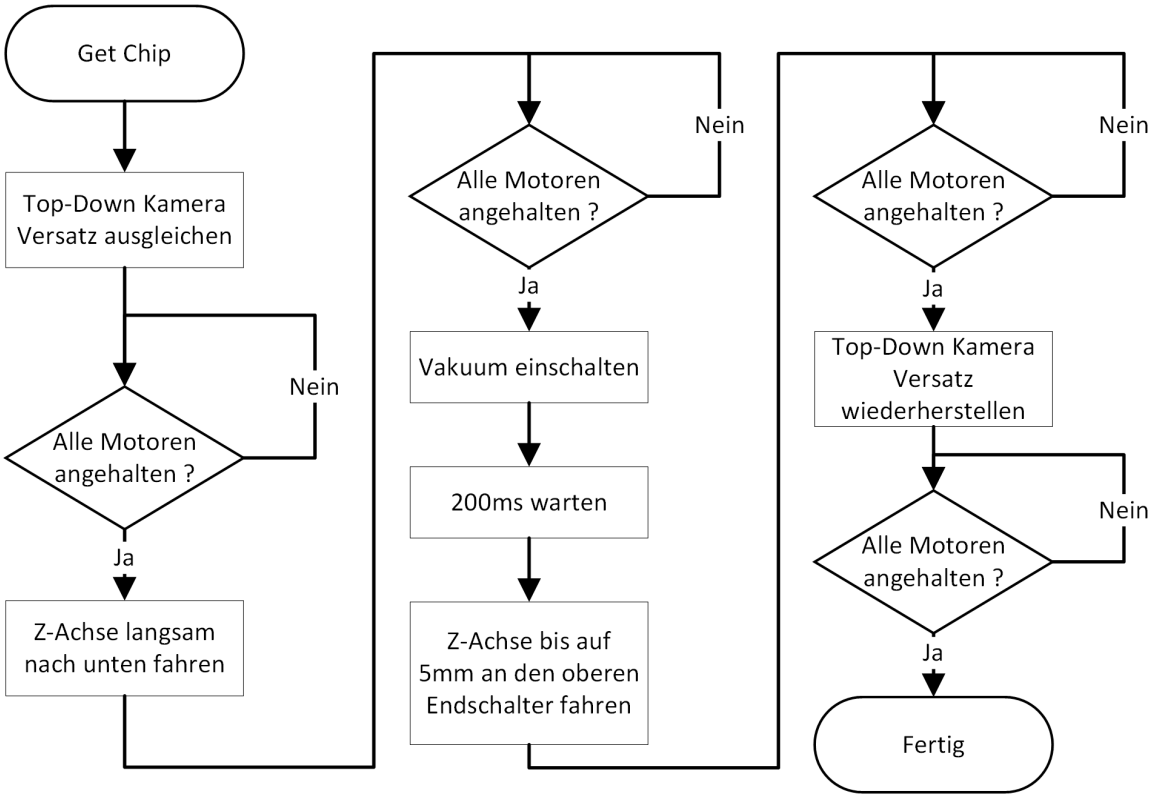


Abbildung 7.6.: Aufnahmeroutine Ablaufdiagramm

7.5. Ablegen des ICs

Das Ablegen eines ICs funktioniert ähnlich wie dessen Anheben. Auch beim Ablegen wird der Versatz zwischen Fahrtkamera und Saugheber ausgeglichen und wiederhergestellt. Grundsätzlich fährt der Saugheber mit dem angesaugten IC nach unten, bis der Endschalter des Saughebers ausgelöst wird, wodurch die Z-Achse stoppt. Dann wird der Unterdruck deaktiviert. Wenn die Z-Achse jetzt direkt wieder hochfahren würde, würden insbesondere ICs mit kleinem Gehäusetyp wieder mit angehoben werden und sich im Laufe der Aufwärtsfahrt lösen. Grund dafür ist, dass der Unterdruck im Saugnapf nicht zeitgleich mit der Ansteuerung des Magnetventils vergeht. Zusätzlich ist der Saugnapf durch sein Material etwas klebrig, wodurch leichte ICs daran kleben bleiben können. Eine Lösung wäre es, anstatt den Unterdruck nur zu deaktivieren, von Unterdruck auf Druckluft umzuschalten, um so den IC von dem Saugheber abzudrücken. Diese Lösung war jedoch aufgrund mangelnder Ausstattung nicht möglich. Stattdessen werden die ICs nach Deaktivierung des Unterdrucks quasi am Tray abgestreift. Wie bei der Aufnahme-Routine gibt es für das Ablegen ebenfalls zwei Funktionen. Die manuelle Funktion ist `on_placeICpushButton_clicked` und die für

die Testroutine ist `placeChip`. Der komplette Ablauf ist in Abb. 7.7 dargestellt und wird im Folgendem beschrieben:

1. Die Voraussetzung ist, dass die Fahrkamera sich mittig über der Stelle befindet, an der der IC abgelegt werden soll.
2. Der Versatz zwischen dem Saugheber und der Kamera wird ausgeglichen und die Z-Achse wird langsam nach unten gefahren.
3. Sobald der untere Endschalter ausgelöst wurde stoppt die Z-Achse und der Unterdruck wird deaktiviert
4. In diesem Zustand wird 1 Sekunde gewartet, damit sich im Saugheber der Umgebungsluftdruck einstellen kann.
5. Die Z-Achse wird langsam um 1 mm nach oben gefahren, um den IC nicht mehr in das Tray zu drücken.
6. Die X-Achse wird um -2 mm verfahren, um so den IC am Tray abzustreifen.
7. Danach wird die Z-Achse wieder bis auf 5 mm an den oberen Endschalter gefahren, die X-Achse um 2 mm und der X- und Y-Versatz negativ verfahren, um die ehemalige Position wiederherzustellen.

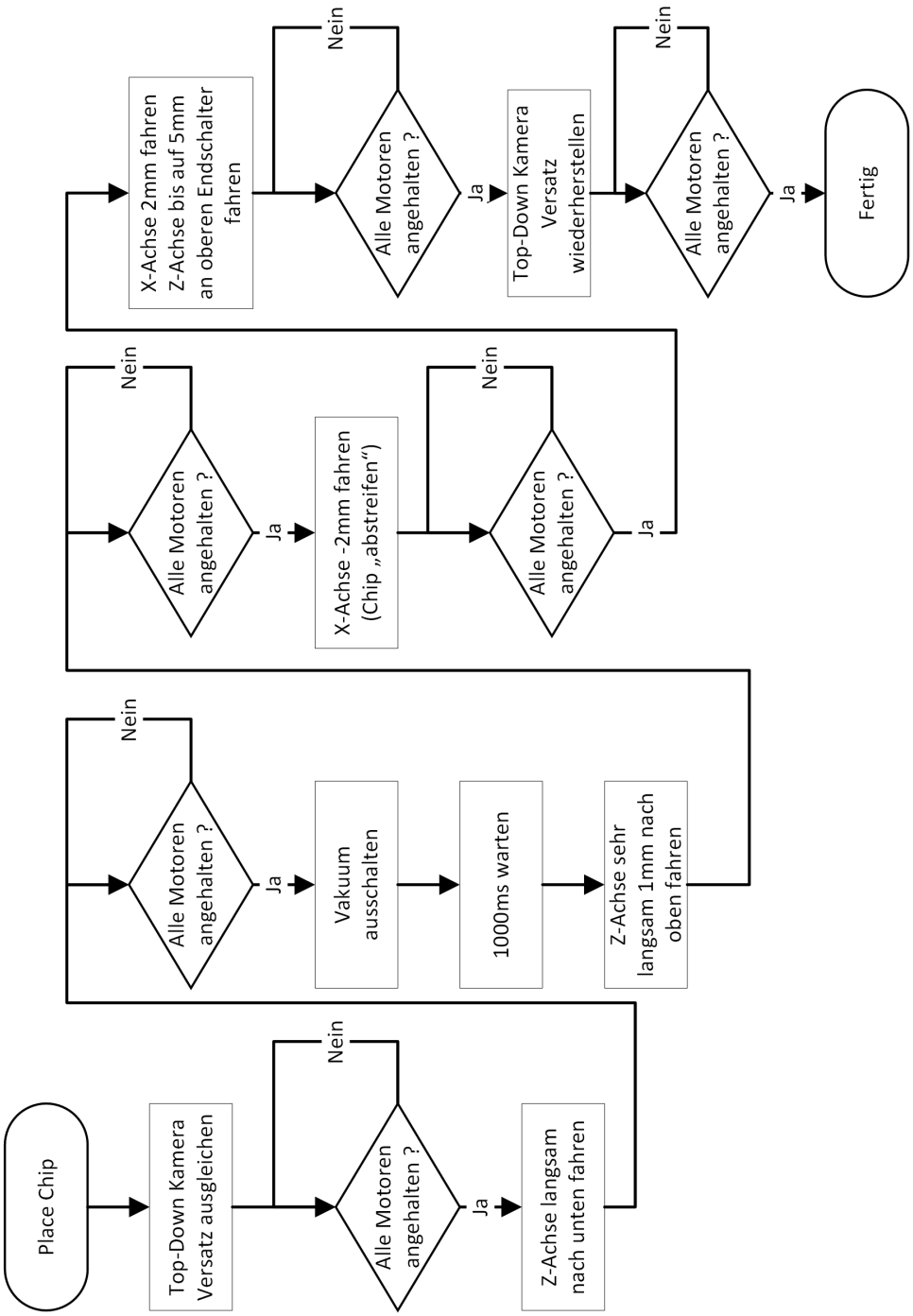


Abbildung 7.7.: Ablegeroutine Ablaufdiagramm

8. Funktionsnachweis und Bewertung

Um zu Überprüfen, ob die entwickelte Software wie geplant funktioniert, wurden mehrere Funktionsnachweise durchgeführt. In diesen wird der Testhandler auf verschieden Funktionen getestet, zum Beispiel auf den Umgang mit Störungen. In den Ablauf wurde nicht eingegriffen, außer es bestand die Möglichkeit einer Beschädigung oder wenn eine Fortsetzung der Tests nicht möglich oder nötig waren.

8.1. Funktionsnachweis der Testroutine

Für den Funktionsnachweis der IC-Testroutine wurde ein komplett gefülltes Tray, mit *TMC5041* ICs im QFN48 7x7 Gehäuse, in die Halterung des Sourcetrays gesetzt. Diese Trays fassen 416 ICs. In die Halterungen des Fail- und Passtrays sind jeweils leere Trays eingesetzt worden. Nach dem Einschalten und Laden der Konfigurationsdateien wurde das Homing durchgeführt und die Kameras kalibriert. Danach wurde die Testroutine gestartet. Nach ca. sechs Stunden musste der Testlauf aus zeitlichen Gründen abgebrochen werden. In dieser Zeit wurden von den 416 vorhandenen ICs des Sourcetrays 379 getestet und sortiert. Davon wurden 301 ICs in das Passtray und 78 ICs in das Failtray sortiert. Das entspricht ungefähr einem Verhältnis von vier zu eins. Dies liegt an der Einstellung des Zufallsgenerators und dessen Auswertung (siehe Listing E.8 in Funktion `changeResultLabel`). Es blieb kein IC im Sourcetray, beispielsweise durch verhaken, im Tray zurück. Weiterhin wurde kein IC fälschlicherweise als leerer Saugheber erkannt, da keine ICs in der unteren linken Ecke der Arbeitsfläche abgelegt wurden. Da das Verteilungsverhältnis von ungefähr vier zu eins dem der Zufallsfunktion des emulierten Testsockels entspricht, ist davon auszugehen, dass die Bilderkennung alle ICs richtig erkannt hat. Wäre ein IC nicht erkannt worden, wäre er direkt in das Failtray gelegt worden. Dann hätte sich jedoch eine andere Aufteilung auf Fail- und Passtray ergeben müssen. Zusätzlich hätte die fehlerhafte Detektion eines ICs auch zu einer fehlerhaften Erkennung von dessen Mittelpunkts geführt. Dadurch wäre der entsprechende IC versetzt in den Testsockel gelegt worden. Dabei hätte er sich mit großer Wahrscheinlichkeit im Testsockel verkantet. Dadurch hätte es eine Lücke im Fail- oder Passtray gegeben und am Ende des Testes hätte der IC noch im Sockel liegen müssen. Die Erkennung der Ausrichtung der ICs funktionierte hingegen nicht in allen Fällen. Wie in Abbildung 8.1 zu sehen ist, wurden im Passtray von den 301 einsortierten ICs neun in der falschen

Orientierung eingesetzt. Das entspricht einer Fehlerquote der Bilderkennung von 3%. Wäre statt der emulierten eine echte Testelektronik verwendet worden, hätte diese die verdreht eingelegten ICs als defekt erkannt, und sie wären in das Failtray einsortiert worden.

Besonders bei dem Failtray machten sich Fehler in der Bahnberechnung bemerkbar. Sie äußerten sich dadurch, dass ICs nicht gerade im Tray lagen, sonder leicht schief und oben auf dem Tray (siehe Abb. 8.2 b). Bei dem Passtray sind die selben Fehler im Ansatz zu sehen, allerdings wesentlich schwächer ausgeprägt (siehe Abb. 8.2 a, linke Reihe). Diese Fehler kommen durch mechanische Ungenauigkeiten des Antriebes (Zahnriemen, Zahnriemenrad) zustande. Diese Fehler stellten sich als linear heraus und können kompensiert werden. Zur Ermittelung des Kompensationsfaktors müssen für jedes Tray eigene Messungen erfolgen. Eine vollständige Kompensation war aus zeitlichen Gründen aber nicht möglich. Abgesehen von diesen Fehlern, funktionierte die Testroutine erwartungsgemäß. Es war innerhalb der getesteten sechs Stunden nicht notwendig, in den Testvorgang einzugreifen.

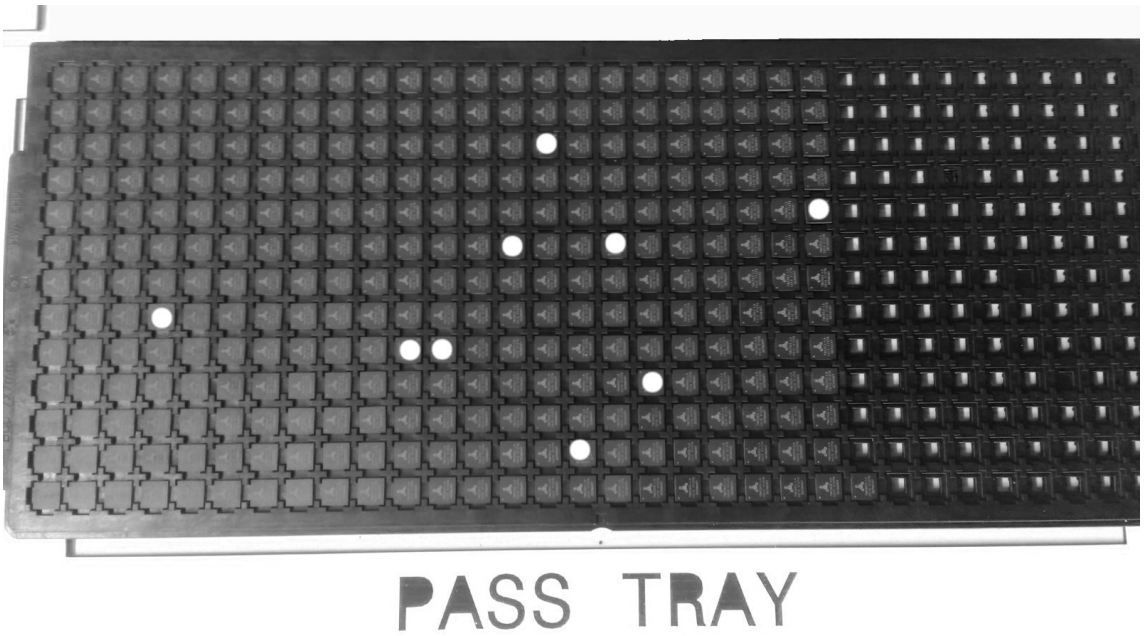
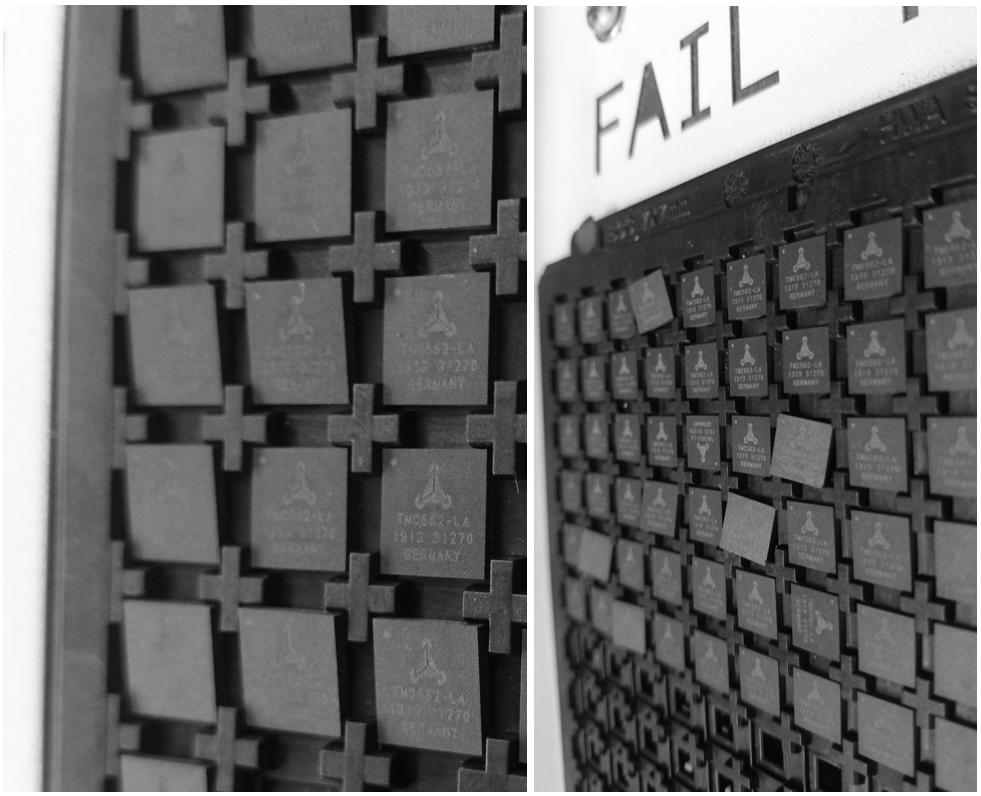


Abbildung 8.1.: Einsortiertes Passtray. Die runden weißen Punkte markieren die falsch orientierten ICs



(a) Passtray (b) Failtray

Abbildung 8.2.: Fehler beim einsortieren der ICs in die Trays

8.2. Funktionsnachweis beim Auftreten von Störungen

Um zu überprüfen, wie kritisch sich bestimmte Störungen auf den Betrieb des Testhandlers auswirken, wurden Störfälle künstlich herbeigeführt und ausgewertet. Im Folgenden sind drei exemplarische Beispiel von auftretenden Störungen beschrieben.

8.2.1. Kein IC angehoben

Diese Störung entsteht, wenn der Saugheber kein IC angehoben hat. Das kann passieren, wenn der IC im Sourcetray festsitzt oder im Sourcetray ein IC fehlt. Dies wurde simuliert indem ein leeres Sourcetray eingelegt und die Testroutine gestartet wurde. In diesem Fall fährt der Saugheber ohne IC über die Tischkamera. Mithilfe dieser wird erkannt, dass kein IC angehoben wurde. Daraufhin fährt der Testhandler in die untere linke Ecke der Arbeitsfläche

und deaktiviert den Unterdruck. Danach fährt er zurück zu dem Sourcetray, um einen neuen IC aufzunehmen. Diese Störung hat keine Auswirkung auf den Testbetrieb, das Fail- und Passtray werden lückenlos aufgefüllt.

8.2.2. Unterbrochene Verbindung zum Computer

Diese Störung entsteht, wenn die Verbindung zwischen der Steuerelektronik und dem steuernden Computer getrennt wird. Das kann zum Beispiel durch ein versehentliches Entfernen des Interface Kabels oder durch einen Absturz oder ein Neustart des Computers hervorgerufen werden. Simuliert wurde dies, indem während der laufenden Testroutine das USB-Kabel zwischen dem Steuermodul und dem Computer entfernt wurde. Der Testhandler hat den letzten empfangenen TMCL-Befehl ausgeführt und sich danach nicht weiter bewegt. In diesem Fall war der letzte empfangene Befehl ein Bewegungsbefehl mit Zielkoordinate. Diese wurde noch angefahren. Wäre zufälligerweise ein TMCL-Befehl ausgeführt worden, bei dem der Testhandler seinen maximal verfahrbaren Raum hätte verlassen müssen, hätten die Motoren durch die Aktivierung der Endschalter angehalten. Der Testhandler nimmt dadurch keinen Schaden und auch die bereits getesteten und einsortierten ICs werden nicht beeinflusst. Allerdings ist eine Wiederaufnahme der Testroutine an der selben Stelle, an der sie unterbrochen wurde, nicht möglich. Das heißt, dass ein zur Hälfte getestetes Tray nicht zu Ende getestet werden kann. Dies stellt einen relativ hohen Aufwand dar.

8.2.3. Saugheber ohne IC in ein Tray abgesenkt

Diese Störung entsteht, wenn die Erkennung eines fehlenden ICs fehlschlägt oder der IC im Testsockel hängen bleibt. Simuliert wurde dies, indem während der Testroutine nach der Erkennung per Tischkamera, der IC manuell entfernt wurde. Dadurch fährt der Saugheber beim Versuch den IC im Tray abzulegen solange nach unten, bis der Endschalter ausgelöst wird. Dafür muss der Saugheber mit etwas Kraft auf etwas aufdrücken. Da für diesen Test ein Tray für QFN48 7 x 7 Gehäuse verwendet wurde, passte der Saugheber gerade durch die leeren Aussparungen des Trays hindurch und hielt erst an der Arbeitsfläche an. Dadurch verhakte sich der Saugheber in dem Tray. Die Folge war, dass bei dem Hochfahren des Saughebers das ganze Tray mit angehoben wurde. Nach einigen Zentimetern löste sich das Tray wieder und fiel zurück auf die Arbeitsfläche. Durch den Aufprall fielen alle, zuvor einsortierten, ICs aus dem Tray. Die Testroutine musste abgebrochen werden. Als Folge müssen alle, aus dem Tray gefallenen, ICs wieder eingesammelt und einsortiert werden. Einige ICs sind auch von der Arbeitsfläche bis auf den Boden gefallen und können dabei Schaden genommen

haben. Der dadurch entstehende Aufwand ist sehr hoch. Allerdings hängt das Auftreten dieses Fehler von dem verwendete Tray ab. In einem Tray für größere Gehäuse hätte sich der Saugheber nicht verhakt, wodurch diese Störung ohne Folgen geblieben wäre.

9. Potentielle Störungen

Während des Betriebs des Testhandler kann es zu verschiedenen Störungen kommen. Im Folgenden soll erläutert werden, welche Störungen auftreten können, welche Folgen sie haben und ob und wie sie verhindert werden können.

9.1. Abgefangene Störungen

In diesem Abschnitt werden beispielhaft einige Störungen vorgestellt, welche von der Software ausgeglichen werden können. Diese Störungen haben keine Auswirkung auf den Betrieb des Testhandlers oder die Testroutine.

Störung	Nicht rechtwinkliger Aufbau des LitePlacers.
Ursache	Durch die günstige Fertigung, sind die Einzelteile des LitePlacers nicht genau genug, um einen annäherungsweise ideal rechtwinkligen Aufbau zu ermöglichen.
Auswirkung	Die X- und Y-Achse stehen in einem unbekannten Winkel zueinander, was eine genaue Bahnberechnung nicht möglich macht.
Lösung	Wie in Kapitel 4.5.1 beschrieben, wird findet ein Ausgleich dieses nicht rechtwinkligen Winkels statt, wodurch eine präzise Bahnsteuerung möglich wird.

Tabelle 9.1.: Nicht rechtwinkliger Aufbau des LitePlacers

Störung	Zufällige manuelle Bedienung während der Testroutine.
Ursache	Während die Testroutine läuft drückt der Benutzer aus Versehen auf eine der manuellen Kontrolltasten.
Auswirkung	Alle Positionen die der Testhandler anfährt stimmen nicht mehr mit den berechneten überein. Die Testroutine muss abgebrochen werden.
Lösung	Alle Schaltflächen, die eine manuelle Kontrolle des Testhandler erlauben, werden während der Testroutine deaktiviert.

Tabelle 9.2.: Zufällige manuelle Bedienung während der Testroutine

Störung	Reflektierende Anschlussflächen des ICs mit daraus resultierender schlechter Bilderkennung.
Ursache	Durch direkte Beleuchtung der Unterseite des ICs kommt es zu Reflexionen in Richtung der Tischkamera.
Auswirkung	Durch die Reflexionen ist das aufgenommene Bild stark überbelichtet. Es können keine Details erkannt werden, wodurch die Bilderkennung nicht genutzt werden kann.
Lösung	Es wurde eine indirekte Beleuchtung entworfen, die ein optimale Ausleuchtung des ICs und damit verwendbare Bilder ermöglicht.

Tabelle 9.3.: Reflektierende Anschlussflächen von ICs

Störung	Saugheber ohne angesaugten IC in das Fail- oder Sourcetray absenken.
Ursache	Wenn ein IC nicht waagerecht oder verklemmt im Sourcetray liegt oder gar nicht vorhanden ist, kann das dazu führen, dass der Saugheber den IC nicht ansaugen kann.
Auswirkung	Es wird mit Tischkamera kein IC erkannt. Nicht erkannte ICs werden regulär in das Failtray gelegt. Ohne angehobenen IC fährt der Saugheber in das Tray. Dies kann zu einem Ruck im Failtray führen, was bereits abgelegte ICs aus ihren Aussparungen werfen kann. Der Testvorgang müsste abgebrochen werden.
Lösung	Über die Tischkamera wird erkannt, ob der Saugheber leer ist. Dadurch kann ein ablegen im Failtray vermieden werden.

Tabelle 9.4.: Leeren Saugheber in Tray abgesenkt

Störung	IC wird mit der falschen Orientierung in den Testsockel gelegt.
Ursache	Durch unterschiedliche Beleuchtungen oder Eigenschaften der IC Unterseite kann es dazu kommen, dass ein IC als korrekt positioniert erkannt wird, obwohl das nicht der Fall ist.
Auswirkung	Der IC wird in falscher Ausrichtung in den Testsockel gelegt.
Lösung	Die Software des Testsockels erkennt, ob der IC korrekt eingelegt wurde. Wird erkannt, dass der IC falsch ausgerichtet wurde, wird eine Fehlermeldung an die Testhandler Software gesendet, die den IC daraufhin im Failtray ablegt.

Tabelle 9.5.: IC mit falscher Orientierung in den Sockel gelegt

Störung	Trays verrutschen während des Testablaufs.
Ursache	Durch ruckartige Bewegungen des Testhandlers oder durch äußere Einwirkungen können die Trays aus ihren, einseitige offenen, Halterungen hinaus rutschen.
Auswirkung	Die reale Position des Trays stimmt nicht mehr mit der, in der Software angenommenen, überein. Dadurch kann keine genaue Positionierung innerhalb der Trays mehr erfolgen. Die Testroutine muss abgebrochen werden.
Lösung	In die Trayhalterungen wurden Aussparungen eingearbeitet, in die ein Riegel eingelegt werden kann, der die Halterung verschließt. Dadurch können sich die Trays nicht mehr aus der Halterung bewegen (siehe Abschnitt B.2.1).

Tabelle 9.6.: Trays verrutschen während des Testablaufs

Störung	Kleine IC-Gehäuse bleiben beim Ablegen nicht richtig im Tray liegen.
Ursache	Da der Saugheber nur über ein Magnetventil zwischen Unterdruck und Umgebungsluftdruck umschalten kann, hört die Sogwirkung am IC-Gehäuse nicht sofort mit dem Umschalten auf. Der IC bleibt noch einen kurzen Zeitraum angesaugt. Zusätzlich ist der aus Gummi bestehende Saugnapf leicht klebrig, wodurch ein sofortiges Abfallen verhindert wird. Bei ICs mit größeren Gehäusen (zum Beispiel BGA144) sorgt deren Eigengewicht dafür, dass sie sich sofort vom Saugheber lösen. Bei kleinen, leichten Gehäusen (zum Beispiel QFN32) reicht das Eigengewicht hingegen nicht aus, um sich sofort zu lösen.
Auswirkung	Wenn der IC in ein Tray abgelegt werden soll, wird er beim Hochfahren der Z-Achse wieder etwas angehoben und fällt dann schief auf das Tray. Ein Einsortieren der ICs ist somit nicht möglich.
Lösung	Der IC wird bei der Ablegeroutine in das Tray abgesenkt. Sobald der Endschalter des Saughebers ausgelöst wird, der IC also in das Tray gedrückt wird, hält der Z-Motor an und der Unterdruck wird abgeschaltet. Dann fährt die Z-Achse sehr langsam 1 mm nach oben und dann -2 mm mit der X-Achse. Dadurch wird der IC quasi abgestreift, und kann sich durch den kurzen Fall von >1 mm in das Tray legen.

Tabelle 9.7.: Kleine ICs bleiben nicht richtig im Tray liegen

9.2. Nicht abgefangene Störungen

Es können nicht alle Störungen vermieden oder kompensiert werden. Dabei wirken sich die unterschiedlichen Störungen verschieden stark aus. Im Folgenden werden verschiedene, beispielhafte Fehler und deren Auswirkungen beschrieben und bewertet, wobei sie aufsteigend in der Stärke ihrer Folgen sortiert sind.

Störung	Kabelbruch
Ursache	Dadurch, dass der Testhandler viele Leitungen bis zum Saugheber mitführen muss, sind diese Kabel vielen Bewegungen ausgesetzt. Obwohl sie in Kabelketten geführt werden und Kantenschutz an scharfkantigen Stellen genutzt wurde, ist ein Kabelbruch durch, zum Beispiel Materialermüdung, nicht auszuschließen.
Auswirkung	Die Auswirkungen hängen davon ab, welche Art von Leitung unterbrochen wurde. Falls es eine Motorleitung ist, kann dieser sich nicht weiterbewegen. Eine Positionierung ist nicht mehr möglich. Dies kann nicht über die Elektronik und Software erkannt werden. Falls eine Leitung der Endschalter unterbrochen wird, hält die entsprechende Achse an. Das liegt daran, dass die verwendeten Endschalter als Öffner geschaltet sind. Das bedeutet, dass sie eine Leerlauf erzeugen, wenn sie ausgelöst werden. Falls das USB-Kabel der Fahrtkamera beschädigt wird, würde das Steuerprogramm erkenne, dass eine der Kameras nicht mehr angeschlossen ist und abbrechen. Dadurch würde die Elektronik den letzten, empfangenen TMCL-Befehl ausführen und danach warten.
Bewertung	Die Wahrscheinlich ist als relativ gering anzusehen. Auch die Schwere der Störung ist recht gering, da nichts beschädigt wird. Allerdings ist die Fehlersuche und Reparatur mit einem sehr grpßen Zeitaufwand verbunden.

Tabelle 9.8.: Auswirkung von Kabelbruch

Störung	Mechanische Blockade der Achsen
Ursache	Durch versehentliches Eingreifen in den laufenden Testhandler, können die Achsen an der Weiterfahrt gehindert werden. Auch eine Blockade der Linearlager durch Verunreinigungen oder Gegenstände ist möglich.
Auswirkung	Die Motoren haben nicht genügend Drehmoment, um die Blockade zu verschieben. Dadurch bewegt sich die Achse nicht mehr weiter. Da die verwendete Elektronik ein Open-Loop System ist, kann dieser Fehler nicht über die Steuerelektronik erkannt werden. Die Position in der Software stimmt nicht mehr mit der realen überein.
Mögliche Lösungen	Nach dem Testen und Einsortieren jedes ICs könnte ein Homing durchgeführt werden, was viel Zeit kostet.
Bewertung	Dies ist eine sehr unwahrscheinliche Störung, da die Bedienenden Personen angewiesen sind, nicht in den laufenden Testhandler einzugreifen. Auch geht von einem Auftreten des Fehlers keine Beschädigung für den Handler oder die Trays aus. Jedoch würden halbvolle beziehungsweise halbleere Trays übrig bleiben, die aktuell nicht weiterverarbeitet werden können.

Tabelle 9.9.: Mechanische Blockade der Achsen

Störung	Es kommen keine Steuersignale mehr an der Steuerelektronik an.
Ursache	Solche Störung kann soft- und hardwarebedingte Ursachen haben. Softwarebedingt wäre zum Beispiel, dass das Betriebssystem des steuernden Computers in einen Ruhezustand geht oder ein Neustart durchführt. Hardwarebedingt könnte sein, dass sich das Interface Kabel löst oder es durch einen Wackelkontakt unterbrochen wird.
Auswirkung	Das verwendete TMCM-6210 Motion Controller Modul besitzt einen eingebauten Mikrocontroller, der die Motortreiber-ICs ansteuert. Wenn keine neuen Befehle mehr von dem Computer kommen, wird der letzte empfangene TMCL-Befehl ausgeführt und danach auf einen neuen gewartet. Falls der letzte Befehl dazu führt, dass der LitePlacer sich außerhalb seines maximal nutzbaren Bereichs bewegen soll, wird aufgrund der Endschalter eine mechanische Beschädigung verhindert. Dies geschieht indem die entsprechenden Motoren deaktiviert werden. Solch eine Störung hätte deshalb einen Abbruch des Testvorgangs, aber keine Beschädigung zur Folge.
Bewertung	Je nach verwendeten Betriebssystem und dessen Einstellungen kann diese Störungen öfter auftretet. Ein hardwarebedingte Ursache ist abhängig von dem Schutz des Interface-Kabels recht unwahrscheinlich. Es ist eine mittelstarke Störung, da das nicht zu Ende getestete Sourcetray noch restliche ICs enthält und eine Wiederaufnahme der Testroutine aktuell nicht möglich ist

Tabelle 9.10.: Verlust von Verbindung zu Computer

Störung	Mechanische Erschütterung des Testhandlers.
Ursache	Durch äußere Einwirkung auf den Testhandler, zum Beispiel durch Anstoßen.
Auswirkung	Die ICs in den Trays liegen nur locker in ihren Vertiefungen. Durch eine Erschütterung können sie aus diesen herausspringen. Erkennbar wäre das durch die Software nicht. Es kann nur erkannt werden, ob kein IC angehoben wurde. Der Test muss abgebrochen und die ICs wieder einsortiert werden.
Bewertung	Die Auswirkungen dieser Störung zu beheben ist mit großem Zeitaufwand verbunden. Es ist eine recht starke Störung. Allerdings ist die Wahrscheinlichkeit, dass jemand im laufenden Betrieb stark genug gegen den Testhandler stößt, sehr gering.

Tabelle 9.11.: Mechanische Erschütterung des Testhandlers

10. Diskussion

10.1. Zusammenfassung

Ziel dieser Arbeit war es, einen automatischen Testroboter für Chip-Produkte zu entwickeln. Dieser sollte ermöglichen, viele ICs mit vertretbarem Zeit- und Personalaufwand zu testen, programmieren und sortieren zu können. Er sollte in Trays bereitgestellte ICs aufnehmen, diese mithilfe optischer Erkennungstechniken erkennen, um sie dann mit ausreichender Präzision in einem Testsockel positionieren zu können. Nach dem Test sollte entsprechend dem Ergebnis eine automatische Einsortierung in Trays erfolgen. Die Testroutine sollte dabei autonom funktionieren können, sodass Mitarbeiter entlastet werden. Eine einfache Bedienbarkeit sowie die Prüfmöglichkeit verschiedener IC-Typen sind zusätzlich erforderlich, um im Betrieb eingesetzt werden zu können. Für die Erkennung und Positionierung der ICs und der Trays wurde eine Bilderkennungssoftware auf Basis der OpenCV Softwarebibliothek entwickelt. Für die Referenzierung im Raum wurden zwei Kameras verwendet, welche sich mithilfe von optischen Markern orientieren. Zusätzlich wurden diese zur Positionierung und Kontrolle der ICs genutzt. Für eine einfachen Bedienung wurde eine grafische Benutzeroberfläche mithilfe des Qt-Frameworks entwickelt und in die TRINAMIC TMCL-IDE implementiert, wodurch ein eine einfache Wartung und Erweiterbarkeit seitens TRINAMIC möglich ist. Der mechanische Aufbau basiert auf dem LitePlacer, einer preiswerten, quelloffenen Bestückungsmaschine, welche für die Anwendung als Testhandler mechanisch modifiziert wurde. Als elektronische Steuerung wurde das TMCM-6210 Modul von TRINAMIC verwendet, wobei innerhalb der Software die Wahlmöglichkeit besteht, auch andere Module von TRINAMIC zu verwenden. Der entwickelte Testhandler ermöglicht eine vollautomatische Überprüfung der Funktionen sowie Sortierung eines vollständig gefüllten IC-Trays. Da jedoch zum Zeitpunkt der Fertigstellung die Software für die Integration der Testelektronik noch nicht verfügbar war, wurden die Testläufe nur mit einer Emulation der Testelektronik durchgeführt. Die entwickelte Bilderkennung erkennt zum Großteil die korrekte Position des Prüflings. Es konnten jedoch aus zeitlichen Gründen nicht genügend Messreihen durchgeführt werden, um eine genaue Erkennungsrate angeben zu können. Die Positionierung im Raum mithilfe optischen Marker erreicht eine ausreichende Genauigkeit. Aufgrund mechanischer Ungenauigkeiten entstehen jedoch Fehler bei der Positionierung im Bereich von wenigen Zehntelmillimetern. Dadurch können nicht alle ICs optimal in die eng tolerierten Trays einsortiert werden. Die Verwen-

dung optischer Erkennung ist für die benötigte Präzision theoretisch möglich. Aufgrund der Toleranzen der verwendeten Mechanik kann diese allerdings praktisch nicht erreicht werden. Im Folgenden sind die wichtigsten, erfolgreich umgesetzten Anforderungen aus der Anforderungsanalyse beschrieben:

- Eine Erkennung der Ausrichtung und Position der ICs erfolgt per Bilderkennung.
- Das Referenzieren über den Trays findet mit Hilfe von optischen Referenzpunkten statt.
- Es wurde eine eicht bedienbare graphische Benutzeroberfläche mit Hilfe der Qt-Softwarebibliotheken entwickelt.
- Für verschiedene IC-Typen können Konfigurationsdateien angelegt werden, welche in die Software geladen werden können.
- Die entwickelte Software ist unabhängig von der verwendete Steuerelektronik. Alle TRINAMIC Module, die von der TMCL-IDE unterstützt werden, können genutzt werden.
- Das zu befüllende Fail- und Passtray wird lückenlos aufgefüllt. Die Ausrichtung der ICs innerhalb der Trays ist dabei an die jeweilige Orientierung des Trays angepasst.
- Es wurde eine Testroutine entwickelt, die nach dem Einrichten des Testhandlers einen vollautomatischen Testbetrieb ermöglicht.
- Die Steuersoftware wurde als Plug-in für die TMCL-IDE entwickelt.
- Alle nichtfunktionalen und technischen Anforderungen wurden erfüllt.

10.1.1. Offene Punkte

Aufgrund der begrenzten Zeit dieser Arbeit war nur eine fokussierte Entwicklung der grundsätzlichen Funktionen möglich. Dabei konnten einige Punkte, die für einen vielseitigeren und einfacheren Einsatz des Testhandlers in der Produktion von TRINAMIC nötig sind, nicht umgesetzt werden. Auf diese wird im Folgenden eingegangen:

- Einmessen der Mechanik:
  Die aktuelle Berechnung für die Motoransteuerung basiert auf den theoretischen Werten der Mechanik. So wird zum Beispiel davon ausgegangen, dass die Zahnriemen sich ideal verhielten, sich demnach nicht Dehnen würden. In der Realität kommen solche Faktoren jedoch zu tragen und führen zu einer Differenz zwischen realen und theoretischen Werten. Dadurch wird eine exakte Präzision verhindert. Die realen Parameter müssten in Versuchen bestimmt werden und die Berechnungen dementsprechend angepasst werden.

- Testen verschiedener IC-Gehäusetypen:
  Für die Entwicklung des Testhandler wurde von ICs in QFN Gehäusen ausgegangen. Die Bilderkennung ist dementsprechend auf ein Merkmales der Unterseite des ICs angewiesen. Falls jedoch ICs mit Gehäusearten wie QFP oder SOP getestet werden sollen, kann diese Art der Erfassung nicht genutzt werden. Grund dafür ist, dass bei diesen Gehäusearten die Orientierung nur anhand der Oberseite des ICs feststellbar ist. Um solche Gehäuse verarbeiten zu können, muss eine andere Methode zur Erkennung der Ausrichtung und Position entwickelt werden.

- Mustervorlage in die Konfigurationsdatei des ICs einbinden:
  Zurzeit müssen für den Test eines IC-Typen zwei Dateien geladen werden. Zum einen die Konfigurationsdatei des ICs und zum anderen ein Bild, welches die Mustervorlage für den Bildvergleich darstellt. Eine Einbindung des Bildes in die Konfigurationsdatei würde eventuelle Anwendungsfehler vermeiden.

- Schutz vor Elektrostatischen Entladungen:
  Halbleiter sind gegenüber Elektrostatischen Entladungen besonders empfindlich. Deshalb ist es unerlässlich, um einen praktischen Einsatz des Testhandlers zu ermöglichen, Vorkehrungen zu treffen, die die zu testenden Bauteile schützt. Eine Beschreibung dieser Vorkehrungen ist im Anhang unter A.5 nachzulesen.

- Sicherheitseinrichtungen:
  Der Aufbau des Testhandler besitzt aktuell noch keine Sicherheitsvorkehrungen wie zum Beispiel einen Notaus-Schalter. Um eine sicherere Benutzung zu ermöglichen wäre dies notwendig.

- Testhandler mechanisch fertigstellen:
  Aktuell befindet sich der Aufbau des testhandler noch in einem Entwicklungszustand. Um ihn für einen Praxiseinsatz nutzen zu können, müssten noch Netzteile oder der Steuernde Computer auf dem Aufbau eingebaut werden, um ein in sich abgeschlossenes System zu erhalten.

- Referenzfahrten während des Testablaufes:
  Die aktuell verwendete IC-Testroutine nutzt die optischen Referenzmarker nur zu beginn des Tests zur einmaligen Referenzierung. Wiederholt Referenzierungen während der Testroutine könnten eine höhere Präzision ermöglichen.

- Einbindung eines realen Testsockels:
  Zum Zeitpunkt der Entwicklung des Testhandlers, war die Software zur Einbindung eines realen Testsockels noch nicht verfügbar. Deshalb wurden die zu testenden ICs in einen elektronisch nicht ausgewerteten Testsockel gelegt. Das Verhalten einer echten

Testelektronik wurde mithilfe von Software emuliert. Sie gab zufällig generierte Testergebnisse zurück, nach welchen sortiert wurde. Die Nutzung eine realen Testelektronik ist für einen praktischen Einsatz unerlässlich.

- Durchführung von Langzeittest:
  Aus zeitlichen Gründen konnte der fertige Aufbau des Testhandler nicht über einen längeren Zeitraum betrieben werden. Dies ist notwendig um zu überprüfen, ob unvorhergesehene Fehler und Störungen auftreten könnten.

10.2. Ausblick

Für weitere Entwicklungen würden sich einige Verbesserungsmöglichkeiten anbieten, die im Folgenden aufgezählt sind:

- Bessere Tray Befestigungen:
  Eine stabilere und präziser gefertigte Halterung für die Trays hätte gegenüber den zurzeit verwendete Schablonen den Vorteil, dass die Trays nicht mehr verrutschen könnten wodurch eine höhere Wiederholgenauigkeit zu erreichen wäre.
- Bessere Kameras:
  Die eingesetzten Kameras sind preiswerte Produkte die nicht für eine präzise Nutzung ausgelegt sind und keine besonders guten Bilder liefern. Speziell für die Bilderkennung entwickelte Kameras könnten die Genauigkeit der Erkennung erhöhen und dadurch eine höhere Präzision bei der Positionierung ermöglichen.
- Verarbeitung mehrerer Trays:
  Zu dem jetzigen Zeitpunkt ist auf der Arbeitsfläche nur die Befestigung von drei Trays vorgesehen. Durch Nutzung mehrere Trays, zum Beispiel für Source- und Passtray, könnte der Testhandler für einen längeren Zeitraum autonom arbeiten.
- Closed Loop System:
  Durch ein CLosed Loop System würden entstandene Schrittverluste erkannt und könnten kompensiert werden. Dadurch ließen sich mechanische Ungenauigkeiten besser ausgleichen.
- Effizientere Bahnberechnung:
  Aktuell benötigt der Testhandler je zu testenden IC relativ viel zeit, um die notwendigen Strecken zu fahren. Dabei werden viele Wege verfahren, die für den Transport nicht unbedingt notwendig wären. Durch eine effizientere Bahnberechnung ließe sich je zu testenden IC viel Zeit sparen.

- Angesaugte ICs mit Druckluft ablegen:
  Wie in Kapitel 7.5 beschrieben, bleiben kleine IC-Gehäuse häufig an dem Saugheber hängen. Eine Lösung wäre die Umschaltung von Unterdruck auf Überdruck beim ablegen, um die IC quasi von dem Saugheber abzustoßen. Dadurch könnte auf die zurzeit Verwendete Methode, die relativ viel Zeit kostet, verzichtet werden.
- Drucksensor am Saugheber:
  Durch die Position der Fahrtkamera ist es nicht möglich zu erkennen, ob der Saugheber eine IC erfolgreich anheben konnte Durch einen Drucksensor in der Unterdruckleitung könnte erkannt werden, ob der Saugheber ein Objekt angesaugt hat (niedriger Luftdruck) oder ob er leer ist (höhere Luftdruck).
- Paralleles Testen mehrere ICs:
  Aktuell wird nur der Test mit einem einzigen Testsockel unterstützt. Durch die Verwendung mehrerer Testsockel ließe sich ein höherer Durchsatz an getesteten ICs erzeugen.
- Markierung der getesteten ICs:
  In kommerziellen Testhandler gibt es häufig die Option, ICs nach dem Testvorgang zu markieren, zum Beispiel mit Farbe oder einem Laser. Eine farbliche Markierung der funktionsfähigen oder der durchgefallenen ICs würden eine Verwechselung der getesteten ICs ausschließen.
- Höhere Geschwindigkeiten und Beschleunigung: Aktuell wird der Testhandler relativ langsam verfahren, um Schrittverluste und Beschädigungen während der Entwicklung zu verhindern. Für einen dauerhaften Betrieb könnte die Geschwindigkeits- und Beschleunigungsparametern noch optimiert werden.

Zusammenfassend lässt sich sagen, dass die Hardware und Software des Testhandlers noch ein großes Potential für weitere Entwicklungen bieten. Da der Testhandler in der Produktion von TRINAMIC verwendet werden soll, ist eine Weiterentwicklung und Optimierung wahrscheinlich.

Literaturverzeichnis

[1] The Qt Company. Qt 5.6 Dokumentation. http://doc.qt.io/ (Eingesehen am 15.07.2016).

[2] Mark Fiala. Designing highly reliable fiducial markers. *IIEEE transactions on pattern analysis and machine intelligence*, 32(7):1317–1318, 2010.

[3] Itseez Inc. OpenCV Version 2.4.11 Dokumentation. http://docs.opencv.org/2.4.11/ (Eingesehen am 15.06.2016).

[4] Ralph Maschotta. *Merkmalslistenbasierte Kreuzkorrelationsmethoden für die medizinische Bildverarbeitung*. PhD thesis, Technische Universität Ilmenau, 2009.

[5] Bradski,Gary Rost and Kaehler,Adrian. *Learning OpenCV*. O'Reilly Media, Inc., 2008. ISBN: 978-0-596-51613-0.

[6] Chesnut, Casey. Augmented Reality with Windows Presentation Foundation. http://www.mperfect.net/wpfAugReal/ (Eingesehen am 04.09.2016).

[7] Cypress Semiconductor. JEDEC Tray QFN28 7 x 7 Datasheet. PDF ist der CD beigefügt.

[8] dnt Drahtlose Nachrichtentechnik GmbH. DigiMicro Profi Mikroskopkamera. http://www.dnt.de/DigiMicro-Profi.2.html (Eingesehen am 10.09.2016).

[9] Egoditor UG. QR-Code Generator. http://www.qrcode-generator.de/ (Eingesehen am 12.09.2016).

[10] JOERNS GmbH. Vacuum Generator JoriVac-II. http://www.joerns-gmbh.com/ (Eingesehen am 10.09.2016).

[11] Kuusama, Juha. LitePlacer Dokumentation. http://www.liteplacer.com/the-machine/ (Eingesehen am 01.07.2016).

[12] Lightcraft Technology. Intersense Introduction. http://www.lightcrafttech.com/support/doc/setting-the-stage/intersense-introduction/ (Eingesehen am 04.09.2016).

[13] Seiko Epson K.K. . EPSON Testing Logic IC Handler NX1032XS. http://global.epson.com/products/handler/products/nx1032xs.html (Eingesehen am 30.08.2016).

[14] TopLine Corporation. Understanding Benefits of JEDEC Trays. http://www.topline.tv/JEDEC_Tray.html/ (Eingesehen am 12.09.2016).

[15] TRINAMIC Motion Control. TMC5041 datasheet. PDF ist der CD beigefügt.

[16] TRINAMIC Motion Control. TMCL Reference and Programming Manual. PDF ist der CD beigefügt.

[17] TRINAMIC Motion Control. TMCM-6210-6211 hardware manual. PDF ist der CD beigefügt.

[18] TRINAMIC Motion Control. TMCM-6210-TMCL firmware manual. PDF ist der CD beigefügt.

[19] Waveshare Electronics. QFN48 TO DIP48, Programmer Adapter. http://www.waveshare.com/qfn48-to-dip48.htm (Eingesehen am 10.09.2016).

[20] Wikipedia . Artikel: Halbleiterwerk. https://de.wikipedia.org/wiki/Halbleiterwerk (Eingesehen am 10.09.2016).

[21] Wikipedia. Artikel: Initialisierungsdatei. https://de.wikipedia.org/wiki/Initialisierungsdatei (Eingesehen am 30.07.2016).

[22] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9:62–66, 1979.

[23] Satoshi Suzuki and Keiichi Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30:32–46, 1985.

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

A. Elektronischer Aufbau

A.1. Übersicht

Der Testhandler besteht aus verschiedenen elektronischer Komponenten, welche sich in folgende Gruppen einteilen lassen:

- Bewegungsregelung
- Testeinheit mit Testsockel
- Kameras
- Beleuchtung
- Unterdruckpumpe
- Stromversorgung

Eine schematische Verschaltung ist in Abbildung 3.2 dargestellt. Im Folgenden soll eine kurze Einleitung über die Funktion der einzelnen Gruppen erfolgen.

A.1.1. Bewegungsregelung

Die Bewegungsregelung steuert alle beweglichen Komponenten des Testhandlers, welche aus vier bipolaren Schrittmotoren und einem Magnetventil bestehen. Sie benötigt eine eigene Spannungsversorgung und ist über eine Schnittstelle an einen Computer angeschlossen, auf dem eine Steuerungssoftware läuft.

Für die Entwicklung des Testhandlers wurde als Bewegungsregelung das TMCM-6210-TMCL Modul von TRINAMIC vorgegeben.

A.1.2. Testeinheit mit Testsockel

Die Testeinheit mit Testsockel beschreibt den Aufbau um einen Integrierten Schaltkreis in seiner Funktion zu überprüfen. Er benötigt, je nach IC-Modell, verschiedene Arten der Spannungsversorgung. Die Testeinheit ist über eine Schnittstelle (häufig RS-232 oder USB) an einen Computer angeschlossen, von dem aus der Testvorgang gesteuert und ausgewertet wird.

Die Testeinheit selber ist nicht Bestandteil dieser Arbeit, sie wird lediglich als Black Box behandelt.

A.1.3. Kameras

Der Testhandler besitzt zwei Kameras. Eine, die Fahrtkamera, ist direkt neben dem Saugheber montiert. Sie schaut nach unten, auf die Arbeitsfläche, und wird mit dem Saugheber mitbewegt. Die Fahrtkamera dient hauptsächlich der Referenzierung anhand von Markern auf der Arbeitsfläche. Als Fahrtkamera wird eine günstige USB-Endoskopkamera mit einer Auflösung von 640 x 480 Pixeln genutzt, welche im Lieferumfang des LitePlacers enthalten war [11].

Die zweite ist die Tischkamera. Sie ist unter der Arbeitsfläche befestigt und schaut über eine Aussparung in der Arbeitsfläche nach oben. Sie wird nicht bewegt, sondern wird über ihre Koordinaten angefahren. Der Saugheber mit angesaugtem IC positioniert sich dabei direkt über ihr. Die Tischkamera dient hauptsächlich zur Ausrichtung und Positionierung der ICs. Als Tischkamera wird eine USB-Mikroskopkamera mit einer Auflösung von 5 Megapixeln verwendet [8]. Um sie unter der Arbeitsfläche befestigen zu können, wurde eine Halterung dafür entworfen und gefertigt (siehe Abschnitt B.3).

Auf die Auswahl und Eigenschaften der Kameras wird im Kapitel 4.2 genauer eingegangen.

A.1.4. Beleuchtung

Um ein konstante Bildqualität der Kameras zu erhalten, ist eine möglichst gleichbleibende und gleichmäßige Beleuchtung notwendig. Das erfolgt bei der Fahrtkamera mithilfe von integrierten LEDs, dessen Helligkeit sich manuell einstellen lässt.

Die Tischkamera besitzt zwar auch eine integrierte Beleuchtung, diese wird allerdings

aufgrund schlechter Ausleuchtung nicht genutzt. Stattdessen wird wird eine ringförmige Anordnung von 12 LEDs genutzt, dessen Helligkeit über eine regelbare Spannungsquelle einstellbar ist.

A.1.5. Unterdruckpumpe

Die Unterdruckpumpe wird direkt an die Netzspannung angeschlossen und läuft durchgehend während des Betriebs. Gesteuert wird der Unterdruck nur über das Magnetventil am Saugheber. Verwendet wird eine JoriVac-II, eine Unterdruckpumpe, die speziell für den Betrieb eines Saughebers zum Positionieren oder Sortieren von SMD Bauteilen entwickelt wurde [10].

A.1.6. Stromversorgung

Die Stromversorgung für den Testhandler lässt sich in drei Kategorien aufteilen:

- Bewegungsteuerung:
  Für das TRINAMIC Motion Controller Modul wird ein 24V DC Netzteil verwendet.
- Beleuchtung:
  Um die Helligkeit der Beleuchtung der Tischkamera einstellen zu können, wird dafür ein regelbares Labornetzteil verwendet.
- Testeinheit:
  Je nach Art des zu testenden ICs werden verschiedenen Arten der Stromversorgung benötigt. Die Netzteile werden auf Hutschienen unter der Arbeitsfläche befestigt, was einen einfachen Wechsel ermöglicht.

A.2. TRINAMIC Motion Controller

Neben der Entwicklung von Motortreiber ICs bietet TRINAMIC auch fertige Module zur Motorsteuerung an. Diese variieren unter Anderem in der Anzahl der ansteuerbaren Motoren, den verwendeten Treiber ICs, dem Interface oder in der Art des anzusteuernden Motors (Entweder Schritt- oder BLDC-Motor). Viele Module verfügen weiterhin über eingebaute Mikrocontroller, sodass eine Nutzung ohne externe Steuereinheit, zum Beispiel in Computer, möglich ist. Alle Module lassen sich über die TMCL-IDE ansteuern und parametrisieren, was die Entwicklung von Produkten auf Basis eines Moduls vereinfacht.

A.2.1. TMCL-IDE

Die TMCL-IDE ist eine Entwicklungsumgebung für TRINAMIC Motion Control Module.

Die wichtigsten Features sind:

- Native Unterstützung für alle aktuellen TRINAMIC Module
- Graphische Darstellung von Position und Geschwindigkeit der Motoren in Echtzeit
- Einfachen Zugriff auf die Register der Treiber ICs
- Gleichzeitige Nutzung mehrerer unterschiedlicher Module
- Editor für TMCL-Programme

Durch die automatisch an das angeschlossene Modul angepassten Bedienoptionen, wird das Einstellen von Parametern erleichtert.

Als Beispiel für so ein "Module Toolßei in Abb. A.1 der "Direct mode"dargestellt. Per Dropdown Menüs lassen sich, für das angeschlossene Modul, die einstellbaren Parameter und Aktionen auswählen und ausführen. Gleichzeitig wird der ausgewählte Befehl und die nach dem Senden erhaltene Antwort im Hexadezimal Format dargestellt.

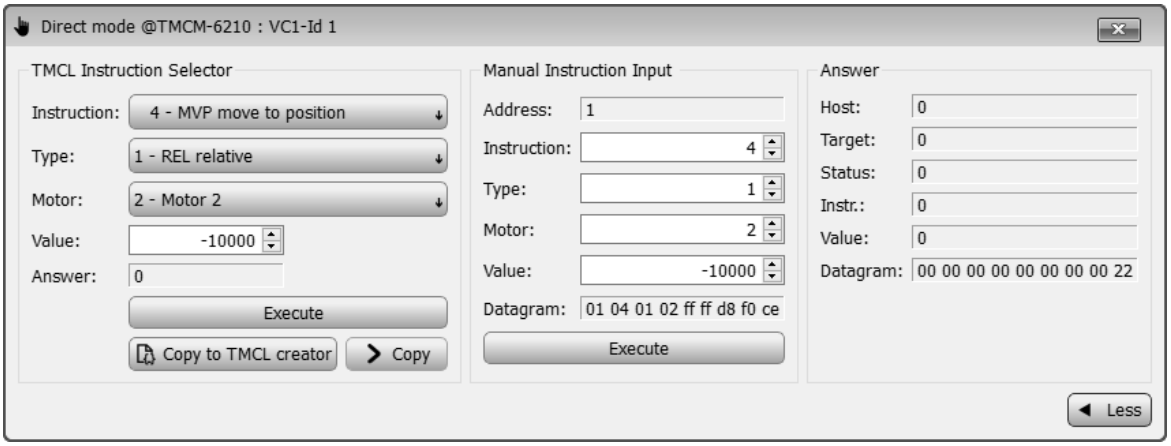


Abbildung A.1.: Direct-Mode von einem TMCM-6210 Modul

Zusätzlich gibt es Plug-ins die weitere, abstraktere Aufgaben übernehmen, zum Beispiel das Updaten der auf einem Modul installierten Firmware. Solch ein Plug-in ist auch die in dieser Bachelor Arbeit entwickelte Steuersoftware für den Testhandler.

A.2.2. TRINAMIC Motion Control Language TMCL

TRINAMIC hat zur Ansteuerung ihrer Module eine eigene Skriptsprache entwickelt, die es auf einfache Weise ermöglicht, Software für die Motion Controller Module zu entwickeln. Je nach Modul kann diese auch auf einem integrierten Mikrocontroller laufen, was eine standalone Anwendung ermöglicht, ohne die Notwendigkeit einer zentralen Steuerung. Dadurch entfällt die Notwendigkeit eine eigene Embedded Firmware zu schreiben.
Die TMCL besteht aus mnemonischen Befehlen, wobei zu beachten ist, dass nicht jedes Modul alle Befehle unterstützt und das bestimmte Befehle nur im Standalone-Mode oder Direct-Mode verwendet werden sollten.
Für eine ausführlichere Erklärung sei an dieser Stelle auf Quelle [16] verwiesen.
Folgende Liste gibt eine Übersicht über die gebräuchlichsten Befehle :

- Motion Commands
  - ROL: Rotate left
  - ROR: Rotate right
  - MVP: Move to position
  - MST: Motor stop
  - RFS: Reference search
  - SCO: Store coordinate
  - CCO: Capture coordinate
  - GCO: Get coordinate
- Parameter Commands
  - SAP: Set axis parameter
  - GAP: Get axis parameter
  - STAP: Store axis parameter into EEPROM
  - RSAP: Restore axis parameter from EEPROM
  - SGP: Set global parameter
  - GGP: Get global parameter
  - STGP: Store global parameter into EEPROM
  - RSGP: Restore global parameter from EEPROM

- Branch Commands.
  This command is intended for use in standalone mode only.
  - JA: Jump always
  - JC: Jump conditional
  - COMP: Compare accumulator with constant value
  - CSUB: Call subroutine
  - RSUB: Return from subroutine
  - WAIT: Wait for a speci1ed event
  - STOP: End of a TMCL-program
- I/O Port Commands
  - SIO: Set output
  - GIO: Get input
  - SAC: Access to external SPI device
- Calculation Commands.
  This command is intended for use in standalone mode only.
  - CALC: Calculate using the accumulator and a constant value
  - CALCX: Calculate using the accumulator and the X register
  - AAP: Copy accumulator to an axis parameter
  - AGP: Copy accumulator to a global parameter
  - ACO: Copy accu to coordinate
- Interrupt Processing Commands.
  This command is intended for use in standalone mode only.
  - EI: Enable interrupt
  - DI: Disable interrupt
  - VECT: Set interrupt vector
  - RETI: Return from interrupt
- Flag Handling Commands
  - CLE: Clear Error Flags

- TMCL Control Commands:
  Weitere spezial Befehle ohne Mnemonic, welche nur für den Direct-Mode (mit externer Steuerung) gedacht sind.

Den mnemonischen Befehlen werden entsprechend ihrer Funktion Parameter mitgegeben, als Beispiel sei hier der `move to position` Befehl erklärt. Dieser Befehl benötigt drei Argumente:

- Die Art der Referenzierung, entweder absolut (ABS) oder relativ (REL). Zusätzlich ist auch die Wahl einer abgespeicherten Koordinate (COORD) möglich.
- Der anzusteuernde Motor. Je nach Modul kann die Anzahl der auswählbaren Motoren schwanken.
- Der zu fahrende Weg in Mikroschritten. Falls eine abgespeicherten Koordinate angefahren werden soll, wird hier die Nummer angegeben, unter der die Koordinate gespeichert wurde.

Im folgenden sind zwei Beispiel Befehle aufgeführt.

Listing A.1: Fahre Motor 2 von der aktuellen Position 10000 Mikroschritte zurück

```
1 MVP REL, 2, -10000
```

Listing A.2: Fahre Motor 0 zu der unter Nr. 8 abgespeicherten Koordinate

```
1 MVP COORD, 0, 8
```

A.2.3. TMCL-Beispiel

In Listing A.3 ist ein kurzes TMCL-Beispiel Programm wiedergegeben. Zu Beginn dieses Programms werden die maximale Geschwindigkeit und Beschleunigung festgelegt. Danach befindet sich das Programm in einer endlos Schleife, in der Motor 0 zwischen 51200 Mikroschritten und -51200 Mikroschritten hin und her fährt.

Listing A.3: Beispiel für TMCL Programm

```
1 SAP 4 , 0 , 51200           // Set max. Velocity
2 SAP 5 , 0 , 51200           // Set max. Acceleration
3
4 Loop:                       // Label
5 MVP ABS , 0 , 512000        // Move to absolute Position 512000
6 WAIT POS, 0 , 0             // Wait until position reached
7 MVP ABS , 0 , -512000       // Move to  absolute Position -512000
8 WAIT POS, 0 , 0             // Wait until position reached
9 JA Loop                     // Jump to label "Loop"
```

A.2.4. TMCM-6210-TMCL Modul

Für die Entwicklung des Testhandlers war die Nutzung des TMCM-6210-TMCL Moduls vorgegeben worden.

Die Hauptmerkmale sind:

- Ansteuerung von bis zu sechs bipolar Schrittmtoren
- bis zu 0.7A RMS / 24V DC je Motor
- Anschlüsse für einen Encoder je Motor
- Anschlüsse für zwei Referenzierungs und einen Homing Schalter pro Motor
- 4x Analog/Digital Input
- 4x Digital Output
- Anschluss für einen Bremschopper Widerstand
- CAN, RS485 und USB Interface

Es kann direkt über die TMCL-IDE und mit der TMCL angesprochen werden und verfügt über genügend I/O Ports und Motorausgänge für den Testhandler. Für weitere Informationen sei auf Dokumentation der Firmware [18] und der Hardware [17] verwiesen.

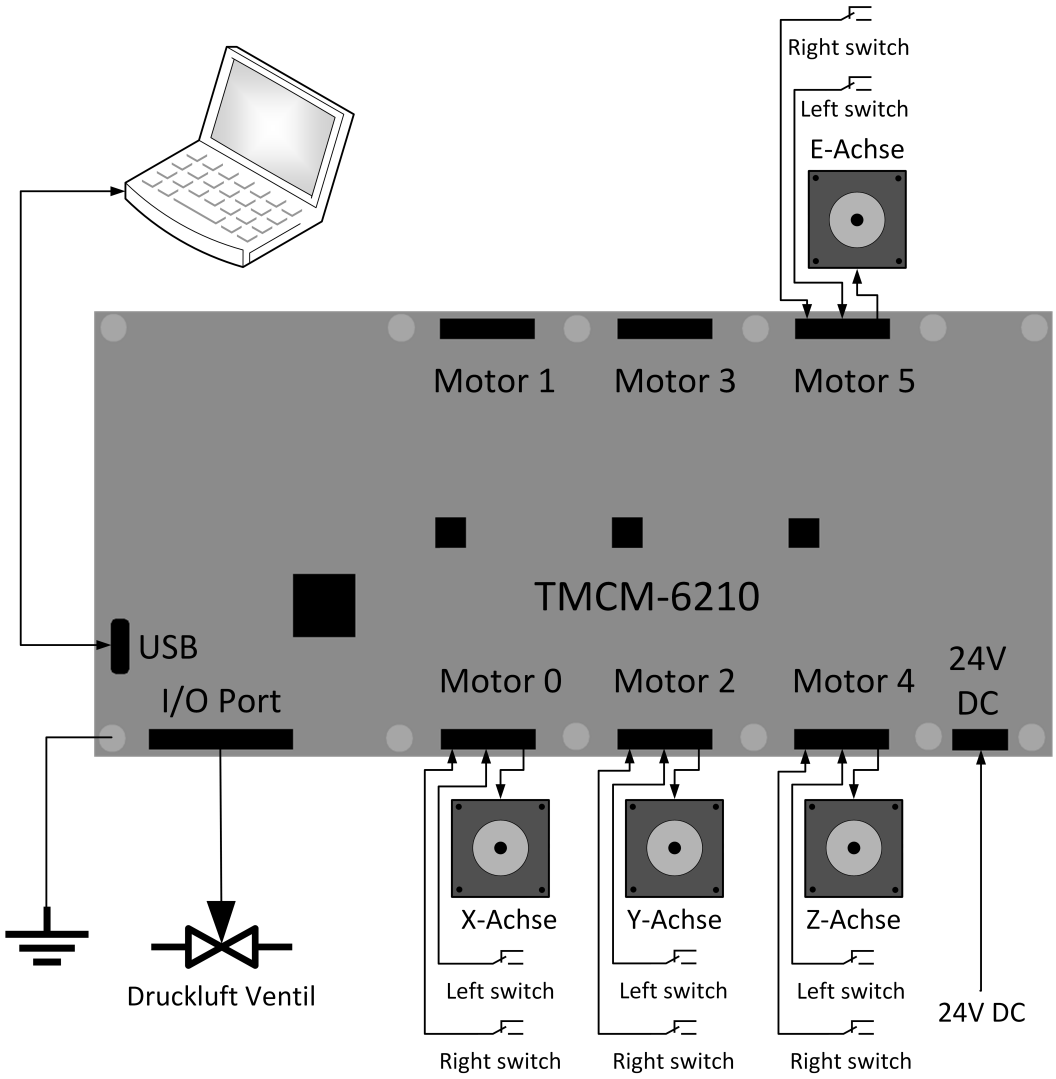


Abbildung A.2.: Schematischer Schaltplan der Steuerelektronik

A.3. Testeinheit

Die Testeinheit setzt sich aus dem mechanischen Sockel und der elektronischen Ansteuer- und Auswerteeinheit zusammen. Beide sind jeweils auf ein bestimmtes IC Modell zugeschnitten, für verschiedene ICs werden verschiedene Testeinheiten benötigt. Die Testelektronik ist dabei an einen Computer angeschlossen, der den Testablauf steuert und die gemessenen Ergebnisse protokolliert.

A.3.1. Testablauf

Für den Test wird ein richtig ausgerichteter IC im Testsockel fixiert, zum Beispiel wie in Abb. A.3 mit einem verriegelbaren Deckel. Danach wird die Testroutine auf dem steuernden Computer gestartet. Dieser sendet vorher festgelegte Stimuli an den IC und wertet dessen Reaktionen aus. Befinden sich diese in einem ebenfalls vorher festgelegten Bereich ist der IC als funktionsfähig einzuordnen. Die Ergebnisse werden dann in einem Testprotokoll gespeichert und der IC aus dem Sockel entfernt und weiterverarbeitet, zum Beispiel sortiert und markiert.

A.3.2. Sockel

Um ein IC auf seine korrekte Funktion überprüfen zu können, ist es notwendig alle Pins kontaktieren um ihn in Betrieb zu nehmen und auszuwerten. Da der IC nach dem Testen weiterverwendet werden soll, ist es nicht möglich ihn dauerhaft zu verbinden, zum Beispiel durch verlöten. Stattdessen gibt es, für jedes Gehäuse spezifische, spezielle Fassungen. Diese ermöglichen eine vollständige Kontaktierung, die schnell zu befestigen und lösen ist. Es gibt verschieden Bauarten der Sockel, die sich in Anzahl der möglichen Testzyklen, Art der Verriegelung, Temperaturfestigkeit und weiteren Eigenschaften unterscheiden. In Abbildung A.3 ist beispielsweise ein Testsockel für ein QFN48 7x7 Gehäuse dargestellt. Die gezeigte Bauform ist für manuelles Testen gedacht, da der Deckel zusätzlich verriegelt werden muss.

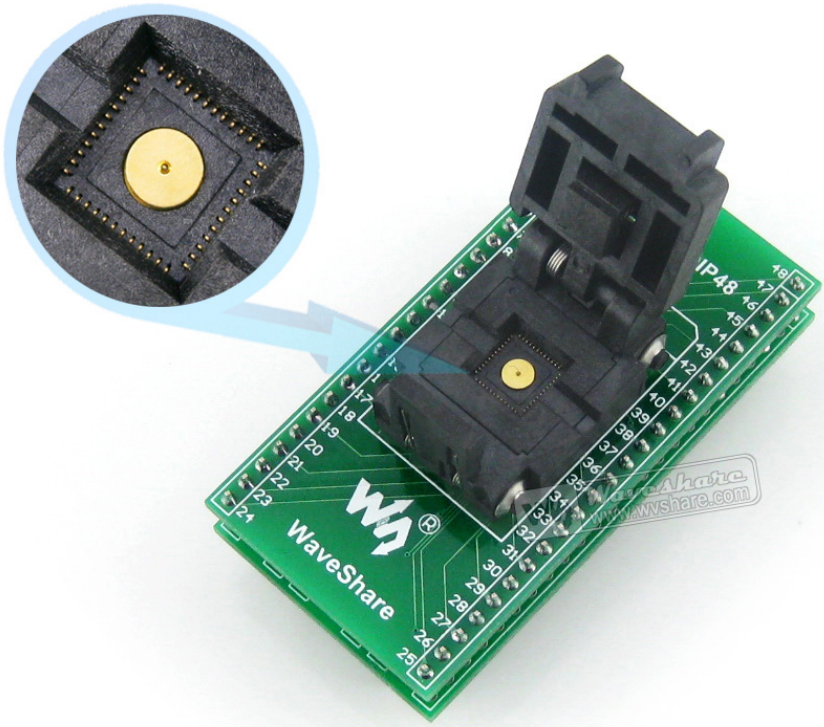


Abbildung A.3.: Testsockel für ein QFN48 7x7 Gehäuse mit Clamshell-Verriegelung für manuelles Testen. Gut zu erkennen sind die einzelnen Pins und der Anschluss der Massefläche in der Mitte [19].

A.3.3. Elektronische Auswertung

Je nach IC Modell müssen verschiedene Arten der Spannungsversorgung und externe Beschaltung, zum Beispiel Filter oder Pufferkondensatoren, an den IC angeschlossen werden, damit er korrekt funktionieren kann. Zusätzlich müssen analoge Signale mithilfe von Analog-Digital-Umsetzern erst digitalisiert werden um sie mit dem Computer auswerten zu können.

A.4. Beleuchtung

Für die Beleuchtung der Tischkamera war es notwendig eine möglichst gleichmäßige, indirekt Beleuchtung der IC Unterseite zu entwickeln. Da die metallenen Pins und Pads der ICs zum Teil stark reflektieren, würde eine direkte, punktförmige Beleuchtung unerwünschte Reflexionen hervorrufen. Das würde die Bilder unbrauchbar machen. Um das zu vermeiden wurde ein LED Leuchtring entwickelt, der die Unterseite vom IC mit 12 weißen LEDs indirekt beleuchtet. Die LEDs sind ringförmig in einen 3D gedruckten Diffusor aus durchsichtigen Kunststoff eingeklebt (siehe Abb. 4.27). Die Konstruktionszeichnung ist in Abschnitt C abgebildet. Durch den Diffusor wird das Licht gestreut und damit gleichmäßiger verteilt, was ein besseres Kamera Bild zur Folge hat.

Die 12 LEDs sind in zwei parallele Stränge aufgeteilt, die aus je sechs LEDs in Reihe bestehen, siehe Abb. A.4. Durch diese Verschaltung ist der Strom, der durch jeweils sechs LEDs fließt, bei allen LEDs gleich, was in eine gleichmäßigere Leuchtstärke resultiert. Um die Helligkeit anpassen zu können, wird der Leuchtring mit einer regelbaren Spannungsquelle gespeist. Für gute Bilder sorgt eine Spannung von ca. 18V.

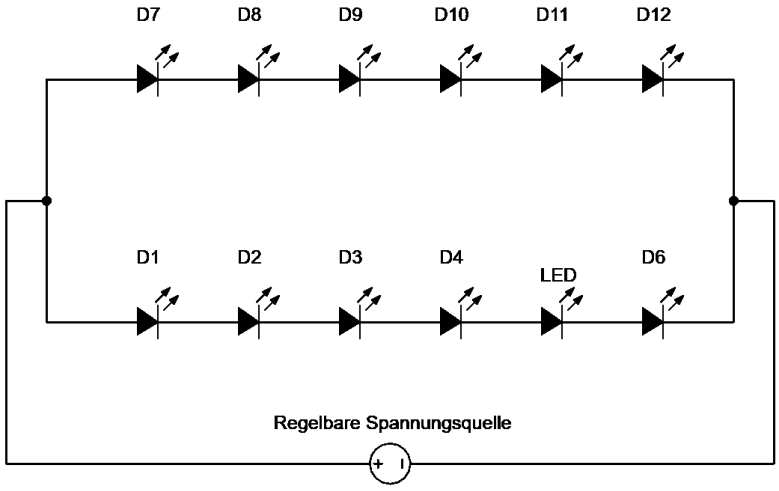


Abbildung A.4.: Verschaltung der 12 LEDs für den Leuchtring

A.5. ESD gerechter Aufbau

Elektrostatische Entladungen (engl. electro static discharge, ESD) sind elektrische Entladungen von zumeist sehr hoher Spannung, bis zu mehreren Kilovolt, die einen hohen Impulsstrom bewirken können. Sie entstehen zumeist durch Reibungselektrizität (triboelektrischer Effekt) oder auch durch Influenz. Beiden Möglichkeiten liegt zu Grunde, dass zwei Leiter von einander isoliert sind und sich zwischen ihnen somit eine elektrische Potentialdifferenz aufbauen kann, welche sich per Funken entladen kann. Integrierte Schaltkreise sind aufgrund ihrer immer kleiner werdenden Strukturgrößen besonders anfällig für Elektrostatische Entladungen. Diese können durch verschieden Arten beschädigt werden:

- Durchschlag von Isolation, wodurch es im Betrieb zu Kurzschlüssen kommen kann
- Durch den hohen Impulsstrom können Teile des ICs schmelzen

In den meisten Fällen ist der IC dann nicht komplett defekt. Er passiert die Endkontrolle und sorgt später im Gebrauch für Störungen und Ausfälle. Um das zu vermeiden ist ein ESD-gerechter Aufbau des Testhandlers nötig.

Aufgrund des mechanischen Aufbaus des LitePlacers sind das X-Portal, Y-Achse und die Z-Achse voneinander isoliert, was Potentialdifferenzen zur Folge haben kann. Der Saugheber ist von dem angesaugten IC durch den Gummi Saugnapf isoliert, der IC könnte sich dadurch aufladen. Die Trays bestehen aus einem Kunststoff der mit Kohlepulver vermischt wurde, wodurch es leitfähig wird und alle ICs in dem Tray das gleiche Potential haben.
Um die ICs vor einer Aufladung zu schützen, müssen die Trays und der LitePlacer das selbe Potential haben. Dafür wird die Tischoberfläche, auf dem die Trays aufliegen mit speziellem Lack elektrisch leitfähig gemacht und mit dem LitePlacer leitend verbunden. Die einzelnen Achsen sind miteinander verbunden und geerdet.Weiterhin ist die Arbeitsfläche geerdet um sich aufbauende Ladungen abzuleiten. Um von außen eingebrachte Ladungen zu vermeiden, sollte der Benutzer, zum Beispiel über ein Antistatikband, geerdet sein.

B. Mechanischer Aufbau

B.1. LitePlacer

Der LitePlacer ist eine quelloffenen Bestückungsmaschine, die für kleinere Unternehmen und Heimanwender entwickelt wurde. Er ist aus Aluminiumprofilen aufgebaut, was ihn einerseits einfach zu modifizieren und preiswert macht, anderseits ist er dadurch nicht besonders stabil. Als Grundlage für den Testhandler wurde der LitePlacer gewählt, da er alle grundlegend benötigten Eigenschaften erfüllt. Dazu zählen:

- Er ist als Flächenportalroboter aufgebaut, was eine einfache Ansteuerung der X-, Y- und Z-Achse als kartesisches Koordinatensystem ermöglicht.
- Er besitzt einen rotierbaren Saugheber.
- Es ist eine Kamera neben dem Saugheber integriert.
- Er verwendet Schrittmotoren, was eine einfache Nutzung der TRINAMIC Komponenten ermöglicht.

Für weitere Informationen über den LitePlacer sei auf [11] verwiesen.

B.1.1. Modifikationen

Der LitePlacer besitzt eine Arbeitsfläche von 350 mm x 510 mm. Für die drei Trays, die Testanlage und die Tischkamera reicht das nicht au, weshalb die X-Achse von 350 mm auf 800 mm erweitert wurde.

B.2. JEDEC Trays

IC-Trays werden weltweit in Produktionsanlagen und anderen Anwendungen verwendet. Wenn jeder Hersteller spezielle eigene Trays benutzen würde, wäre es nicht möglich, universell einsetzbare Maschinen zu fertigen, da sie für jedes neue Tray angepasst werden

müsste. Genormte Trays bedeuten, dass eine Anlage nur auf einen Typ von Tray angepasst sein muss, wodurch sich einfachere und damit günstigere Anlagen möglich wären. Aus diesem Grund wurde von der *JEDEC Solid State Technology Association*, einer Organisation zur Standardisierung von Halbleitern, das JEDEC-Matrix-IC-Tray entwickelt. Es hat genau definierte Maße und Eigenschaften und hat sich in der Industrie durchgesetzt. Einige Spezifikationen der JEDEC Trays werden im Folgenden beschrieben. Für mehr Informationen sei auf [14] verwiesen.

- Die Breite des Tray beträgt 135,9 mm
- Die Höhe des Tray beträgt 315 mm
- Eine Ecke des Trays ist besitzt eine Abschrägung. Diese bestimmt die Ausrichtung des Trays. Die abgeschrägte Ecke ist der Nullpunkt des Trays.

Die Positionen der ICs sind über folgende Parameter angegeben:

- X- und Y-Abstand der ersten Reihe beziehungsweise der ersten Spalte zu der entsprechend parallelen Außenkante.
- X- und Y-Abstand zwischen den ICs

Dadurch kann die Position der ICs relativ zur Außenkante bestimmt werden. Für eine Veranschaulichung sei an diese Stelle auf [7] verwiesen.

B.2.1. Halterung der Trays

Für eine einfache Nutzung des Testhandlers ist es notwendig, dass die Trays einfach gewechselt werden können. Dabei ist es wichtig, dass die Trays beim wechseln nicht zu schräg gehalten werden müssen oder, dass es keine engen Stellen gibt. Das könnte sonst dazu führen, dass beim einlegen oder rausnehmen ICs aus dem Tray fallen könnten. Dafür wurden spezielle Trayhalterungen entworfen, welche so aufgebaut sind, dass die Trays von der Seite in sie eingelegt werden können ohne, dass das droht zu schräg zu sein. Dabei wurde die Form dieser Halterung so gewählt, dass die Trays nur an den Ecken gehalten werden, um so Toleranzen längs der Trays und der Halterung auszugleichen. Es sind auf beiden Seiten der Öffnung, auf Höhe der Tray Außenkante, 1 cm breite Aussparungen einarbeitet, die durch das einlegen eines Riegels das Tray in der Halterung fixiert und es so vor dem verrutschen bewahrt. Zusätzlich sind die Referenzmarker für die Trays in die Oberfläche eingearbeitet worden, um den Abstand zwischen Tray-Außenkante und Marker so konstant wie möglich zu halten. In die Halterung des Failtrays ist zusätzlich noch ein herausnehmbares Element integriert wurden, das *Camera Calibration Piece*. Es besitzt einen Marker und wird vor einer Testroutine dazu genutzt, die Fehlstellung der Tischkamera auszugleichen. Dafür wird das

Camera Calibration Piece mit dem Marker in Richtung Tischkamera über diese gelegt und an dem LitePlacer ausgerichtet. Dann erfolgt die Korrektur des Rotationswinkel (siehe Abschnitt 4.4). Die Halterungen wurden per Laser aus einem bestimmten Material geschnitten, welches normalerweise matt-weiß, nach dem Lasergravieren jedoch matt-schwarz ist. Das bietet den hohen Kontrast, der für die Bilderkennung benötigt wird. Die Konstruktionszeichnungen sind im Abschnitt C abgebildet, wobei diese nur der Anschauung dienen und keinen Anspruch auf vollständige Bemaßungen erheben.

B.3. Tischkamera Halterung

Um die USB-Mikroskopkamera unterhalb der Arbeitsfläche zu befestigen, wurde eine Halterung konstruiert und 3D gedruckt. Diese ermöglicht eine Einstellung der Rotation der Kamera um die Z-Achse und verhindert, dass störendes Licht von der Seite in die Kamera gelangt. In Abbildung B.1 ist die eingebaute Halterung dargestellt. Die Konstruktionszeichnungen sind im Abschnit C abgebildet, wobei diese nur der Anschauung dienen und keinen Anspruch auf vollständige Bemaßungen erheben.

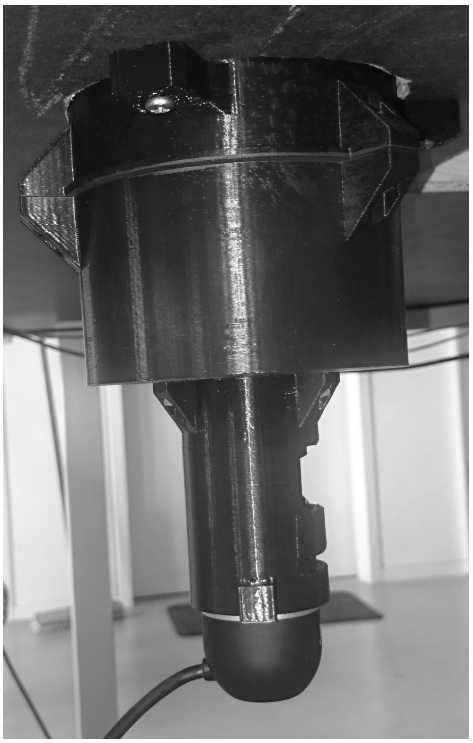
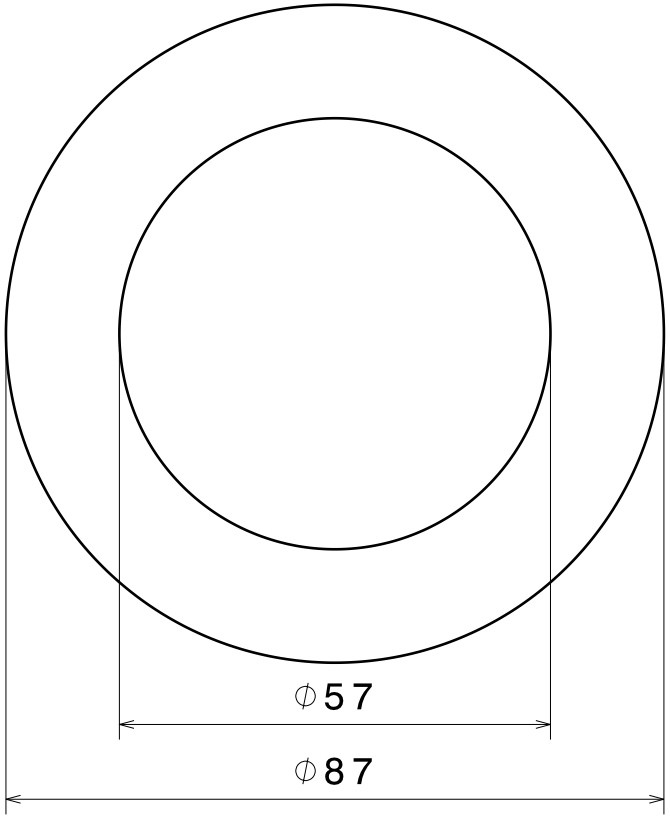


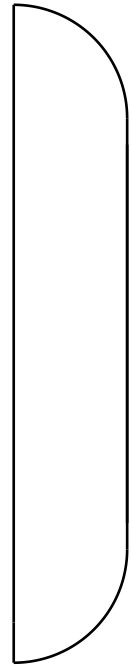
Abbildung B.1.: Eingebaute Halterung der Tischkamera

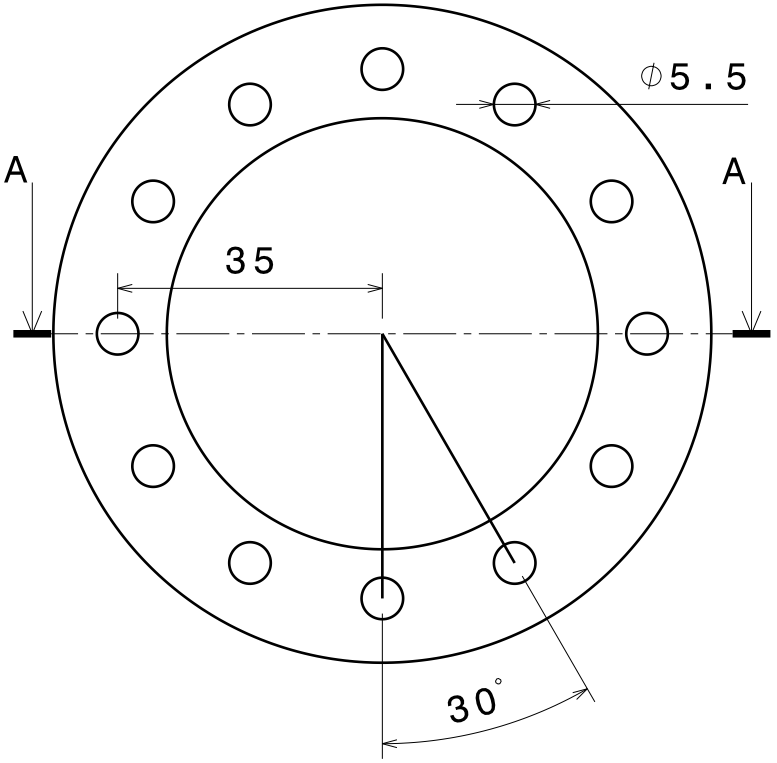
C. Konstruktionszeichnungen

Folgende Konstruktionszeichnungen sind auf den nächsten Seiten abgebildet:

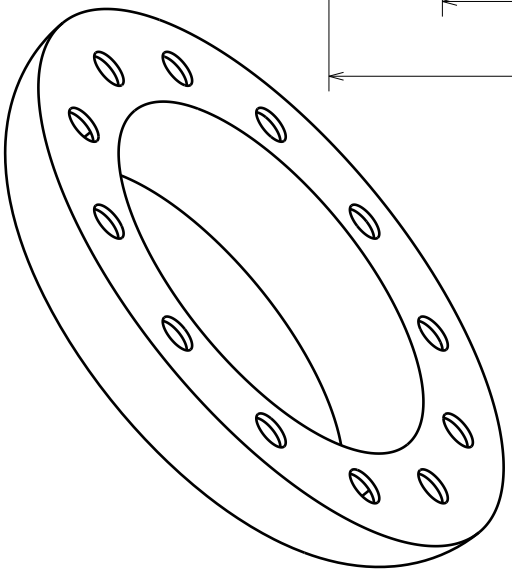
1. Der Diffusor für den LED-Leuchtring der Tischkamera
2. Die Halterung für das Failtray
3. Die Halterung für das Passtray
4. Die Halterung für das Sourcetray
5. Teil 1 der Tischkamerahalterung
6. Teil 2 der Tischkamerahalterung
7. Teil 3 der Tischkamerahalterung
8. Die Zusammenbauzeichnung der Tischkamerahalterung



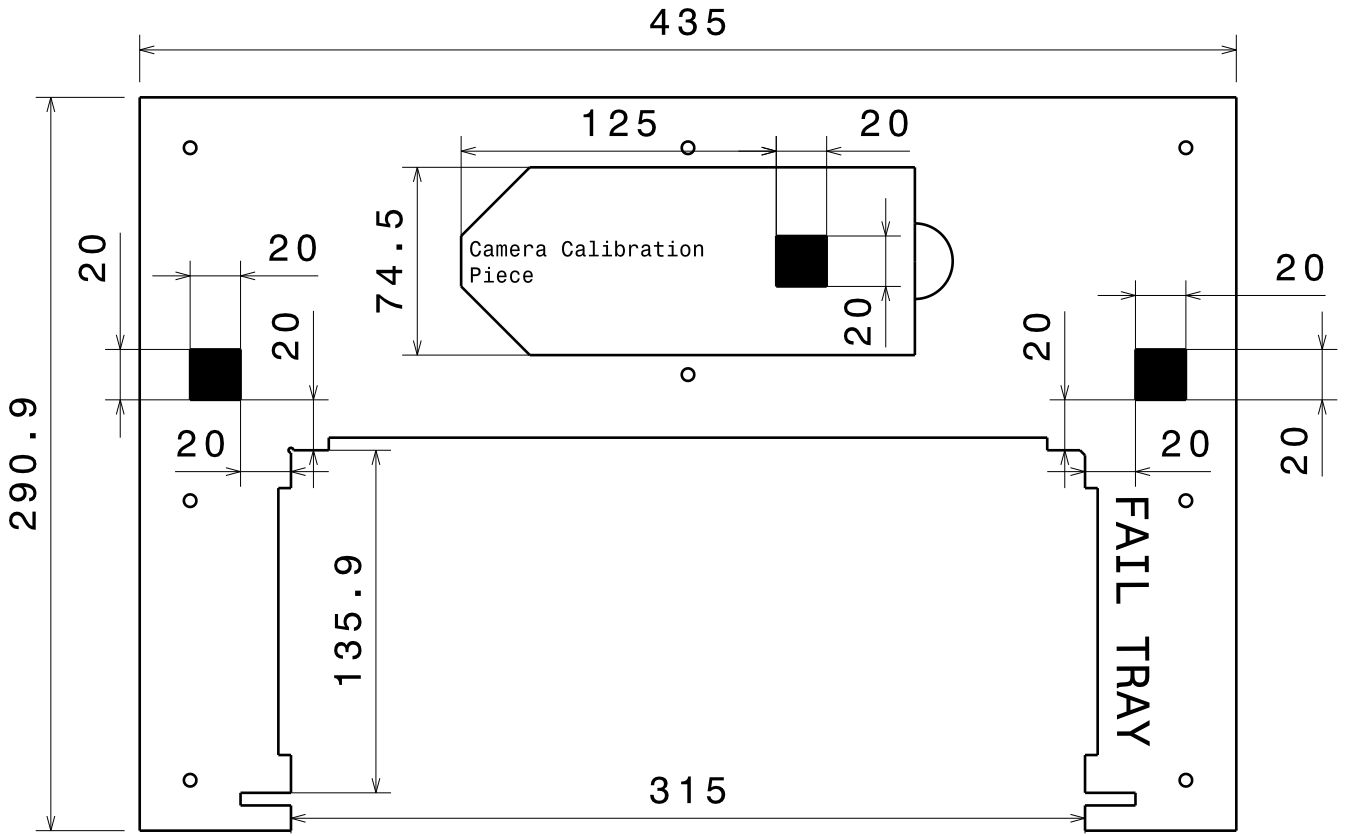




Schnitt A-A

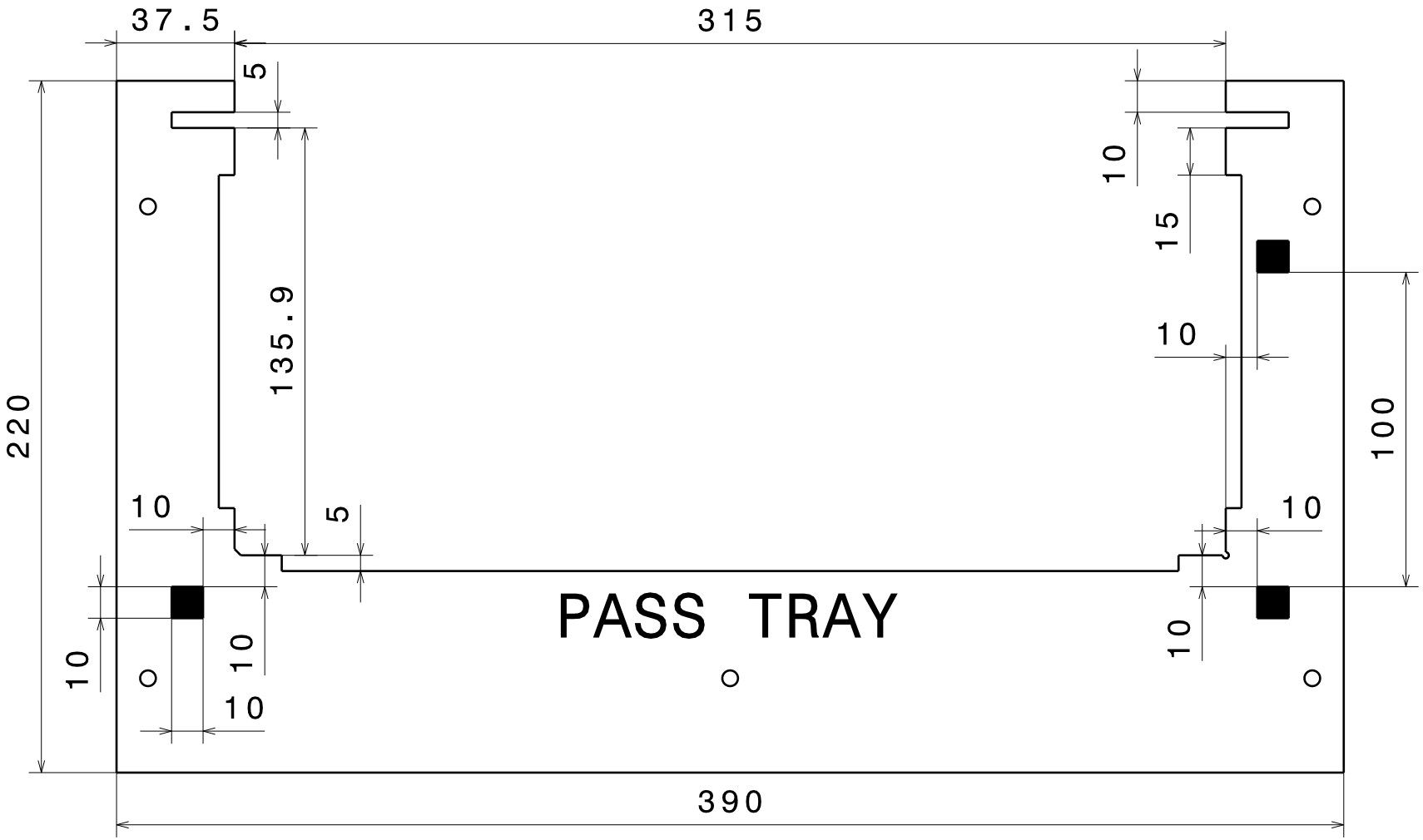


				Werkstoff: PLA	
		Name	Datum	TRINAMIC	
	gezeichnet	Patrick Mattich	24.08.2016	Zeichnungsname: LED Leuchtring mit Diffusor	
Projektion	Maßstab 1:1				
		Alle Bemaßungen in mm		Zeichnungs-Nr.: 1	Index: 01



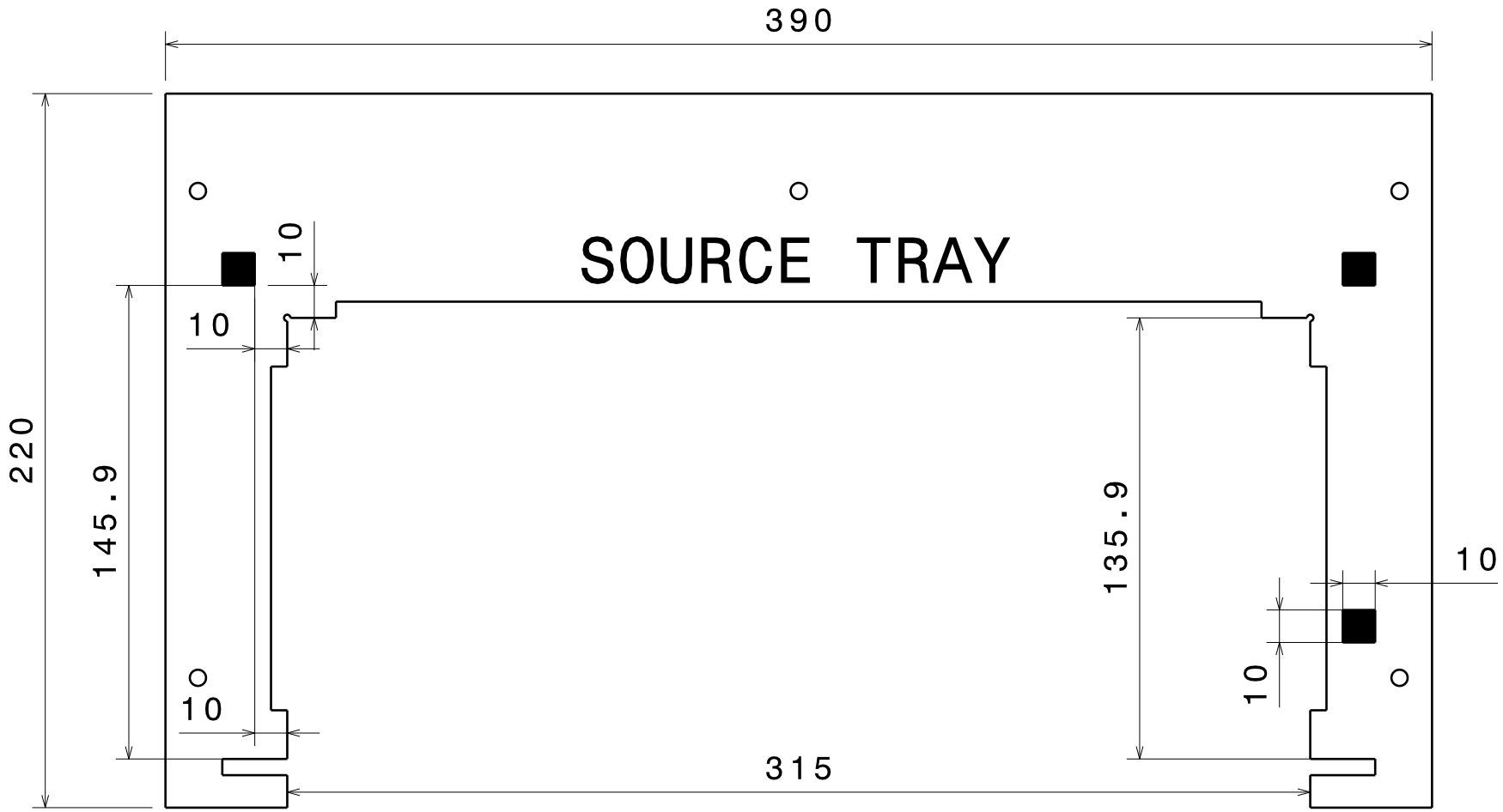
Vorderansicht
Maßstab: 1:3

				Werkstoff: TroLase
Es wurden nur zur Nutzung wichtige Maße eingezeichnet		Name	Datum	TRINAMIC
	gezeichnet	Patrick Mattich	05.08.2016	Zeichnungsname: Failtray Halterung
Projektion	Maßstab 1:3			
		Alle Bemaßungen in mm		Zeichnungs-Nr.: 3 Index: 01



Vorderansicht
Maßstab: 1:2

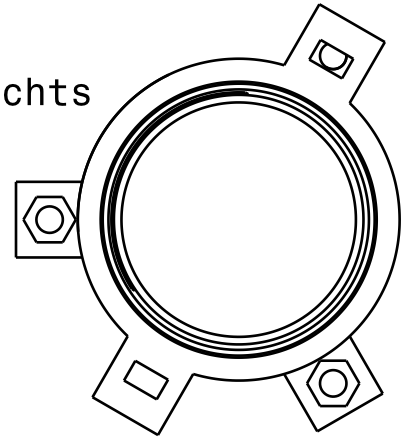
				Werkstoff = TroLase
Es wurden nur zur Nutzung wichtige Maße eingezeichnet		Name	Datum	TRINAMIC
	gezeichnet	Patrick Mattich	05.08.2016	Zeichnungsname: Passtray Halterung
Projektion	Maßstab 1:2			
		Alle Bemaßungen in mm		Zeichnungs-Nr.: 2 Index: 01

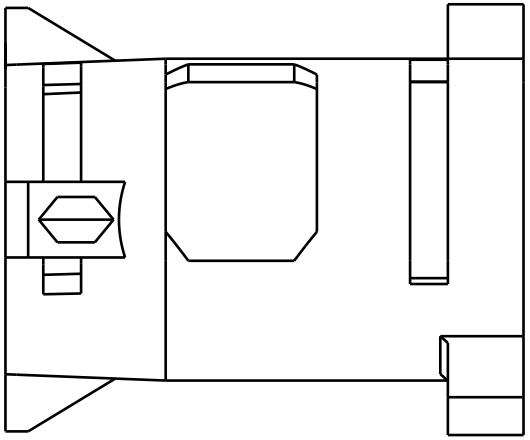


Vorderansicht
Maßstab: 1:2

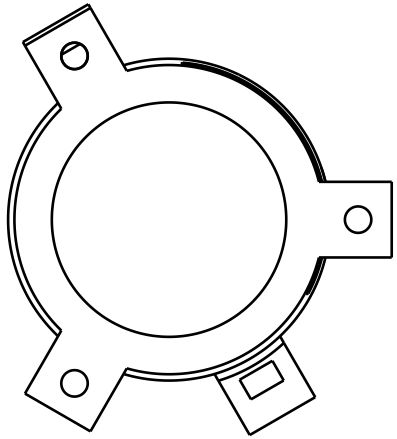
				Werkstoff: TroLase
Es wurden nur zur Nutzung wichtige Maße eingezeichnet		Name	Datum	TRINAMIC
	gezeichnet	Patrick Mattich	05.08.2016	Zeichnungsname: Sourcetray Halterung
Projektion	Maßstab 1:2			
		Alle Bemaßungen in mm		Zeichnungs-Nr.: 4 / Index: 01

Seitenansicht rechts

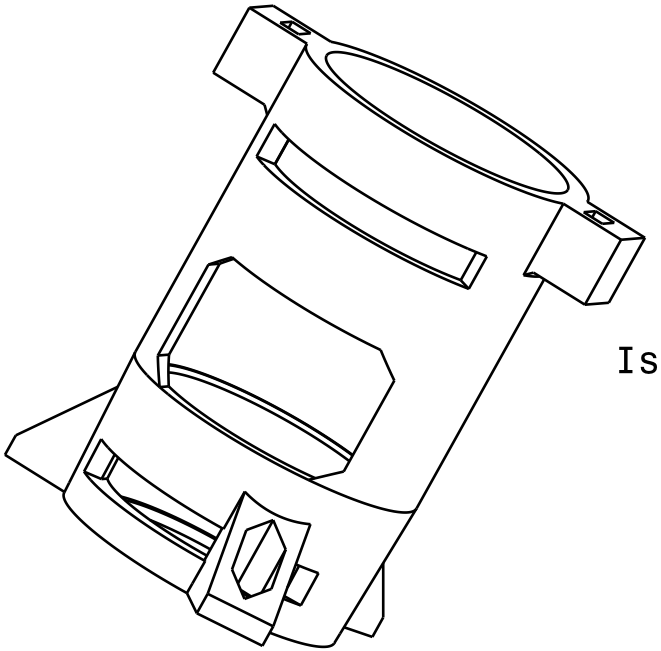




Vorderansicht

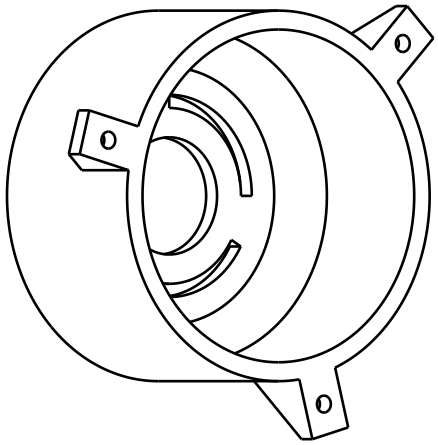


Seitenansicht links



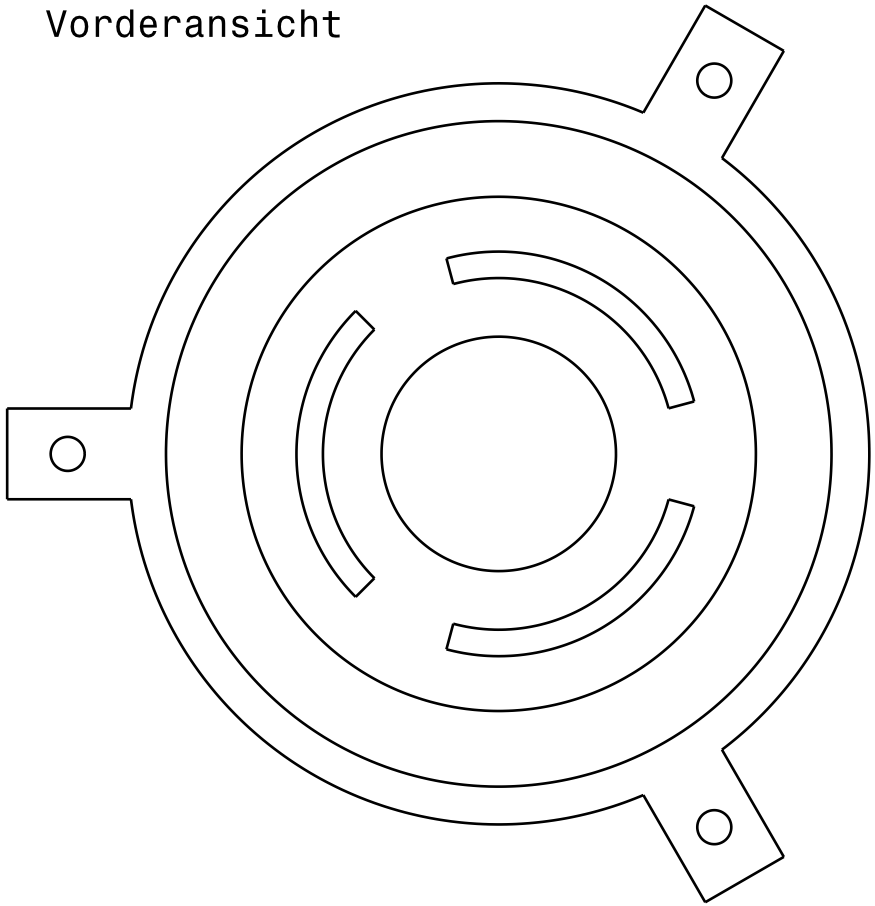
Isometrische Ansicht

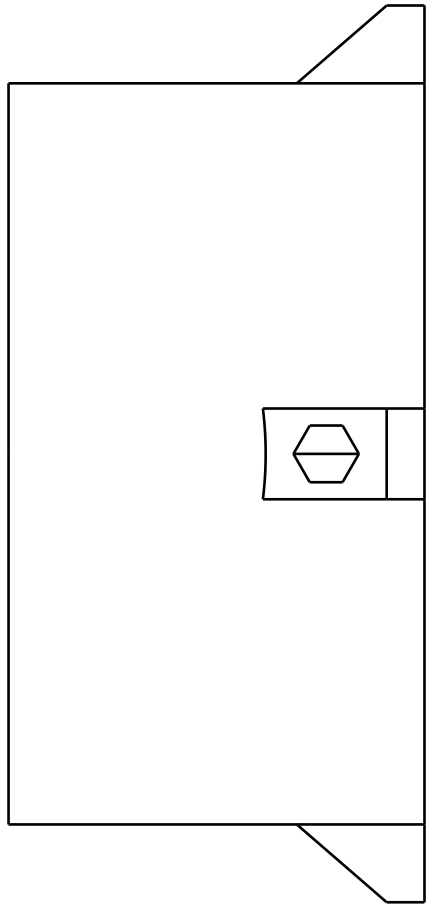
	Benennung			Werkstoff: PLA	
Diese Zeichnung soll nur der Veranschaulichung dienen		Name	Datum	TRINAMIC	
	gezeichnet	Patrick Mattich	12.07.2016	Zeichnungsname: Tischkamera Halterung Teil 1	
Projektion	Maßstab 1:1				
				Zeichnungs-Nr.: 5	Index: 01



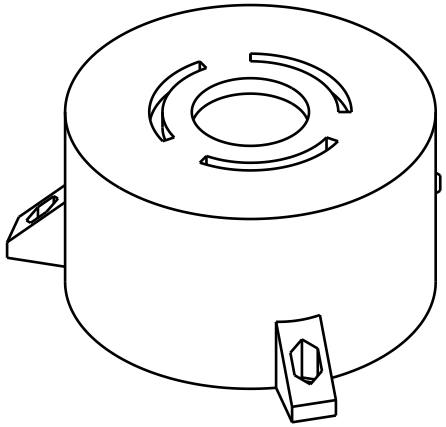
Isometrische Ansicht
Maßstab: 1:2

Vorderansicht



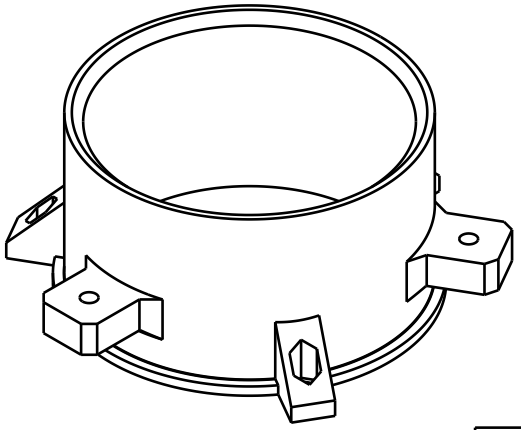


Seitenansicht links

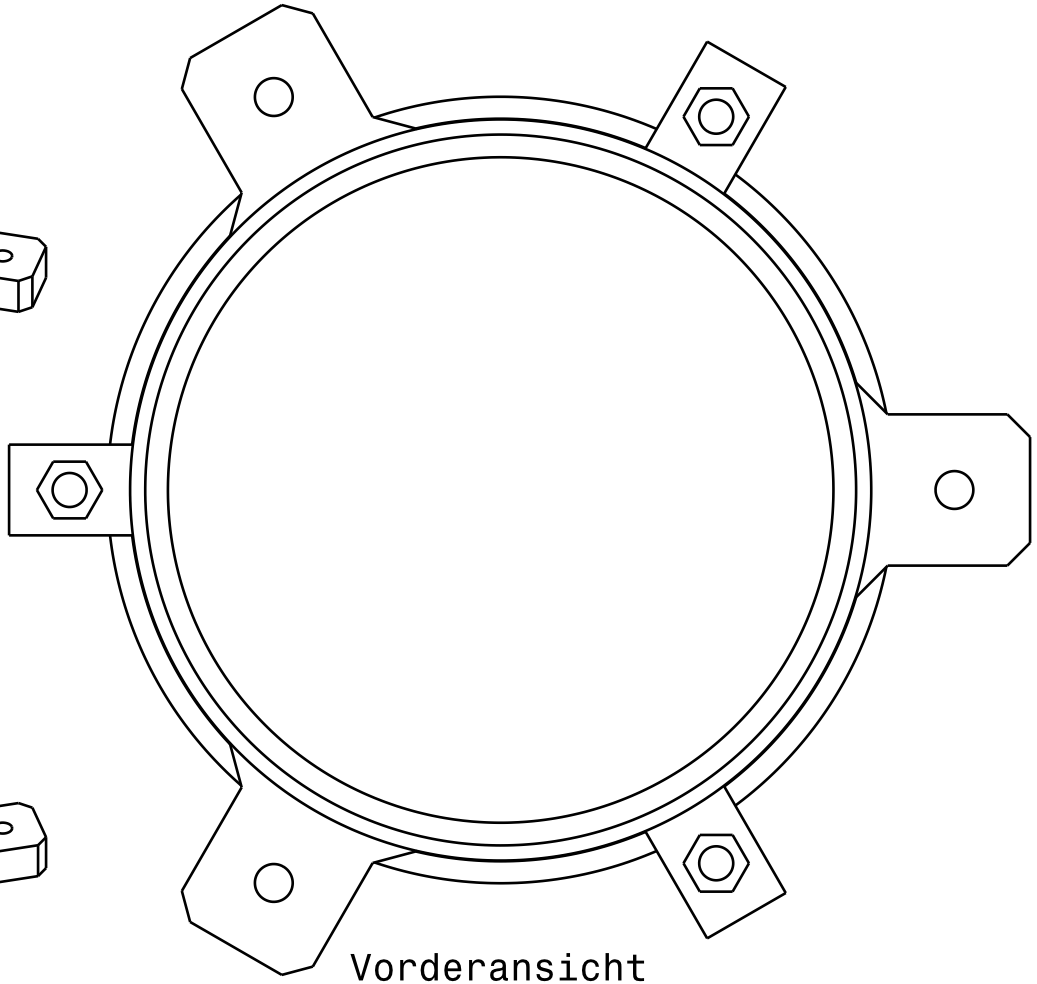


Isometrische Ansicht
Maßstab: 1:2

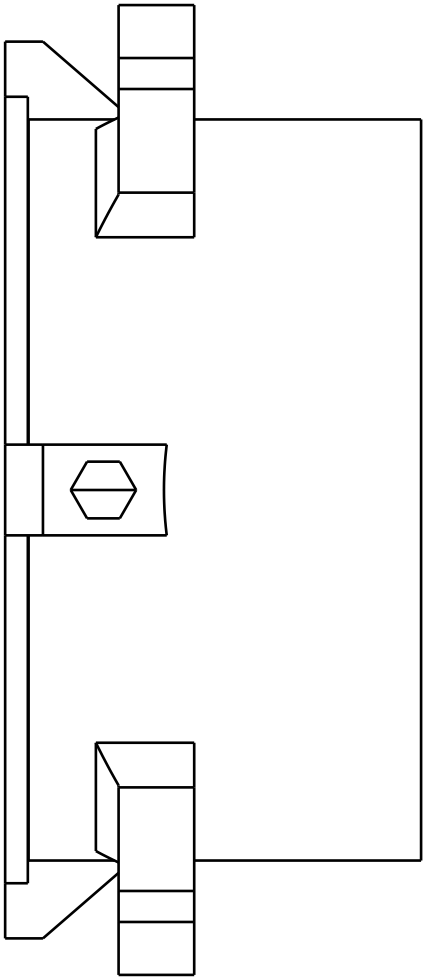
	Benennung			Werkstoff: PLA	
Diese Zeichnung soll nur der Veranschaulichung dienen		Name	Datum	TRINAMIC	
	gezeichnet	Patrick Mattich	12.07.2016	Zeichnungsname: Tischkamera Halterung Teil 2	
Projektion	Maßstab 1:1				
				Zeichnungs-Nr.: 6	Index: 01



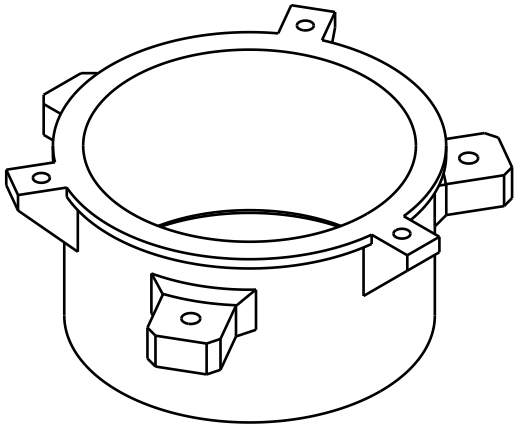
Isometrische Ansicht
Maßstab: 1:2



Vorderansicht



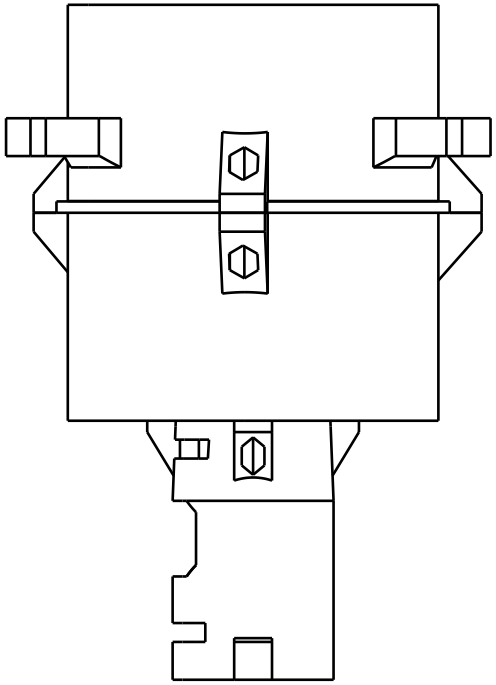
Seitenansicht links



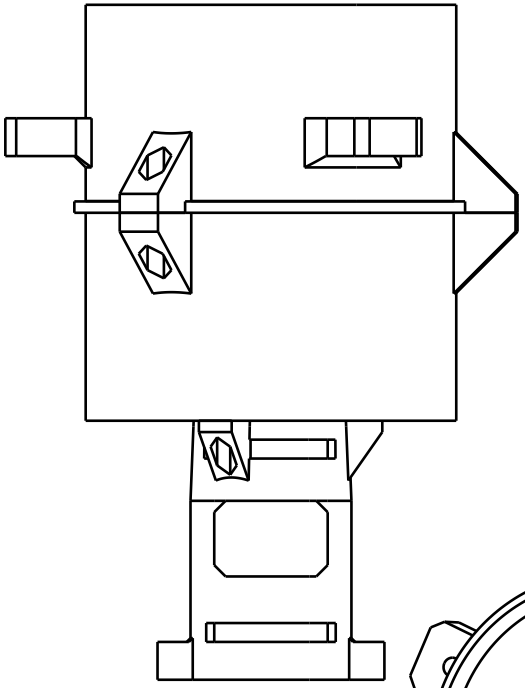
Isometrische Ansicht
Maßstab: 1:2

				Werkstoff: PLA	
Diese Zeichnung soll nur der Veranschaulichung dienen		Name	Datum	TRINAMIC	
	gezeichnet	Patrick Mattich	12.07.2016	Zeichnungsname: Tischkamera Halterung Teil 3	
Projektion	Maßstab 1:1				
				Zeichnungs-Nr.: 7	Index: 01

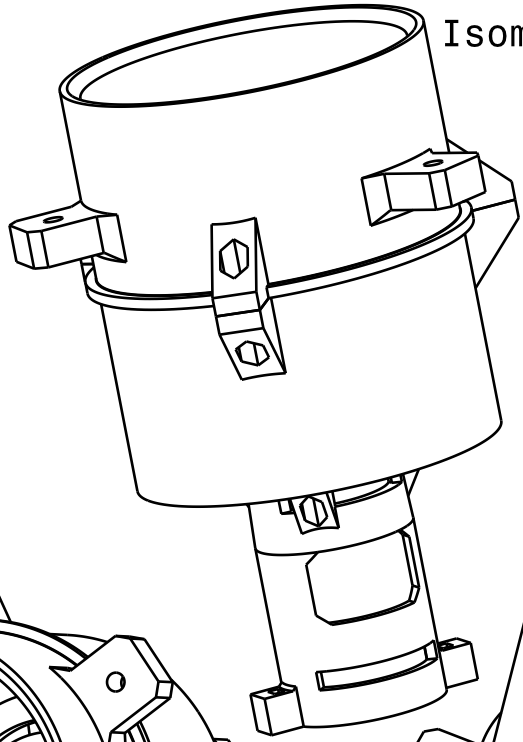
Vorderansicht

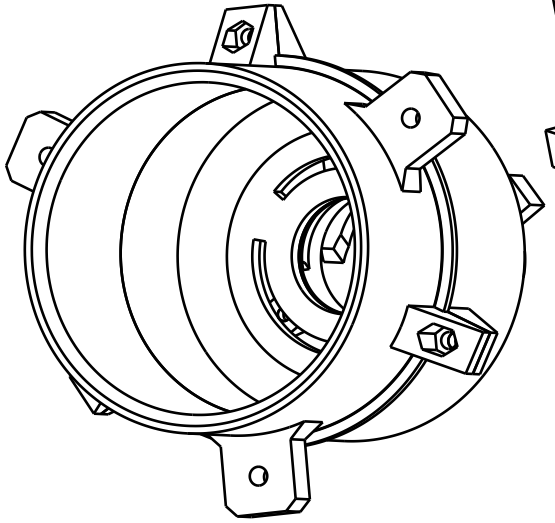


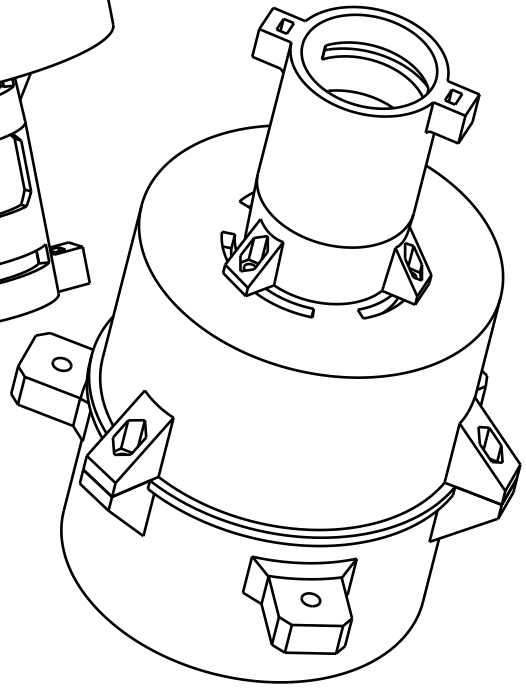
Seitenansicht links



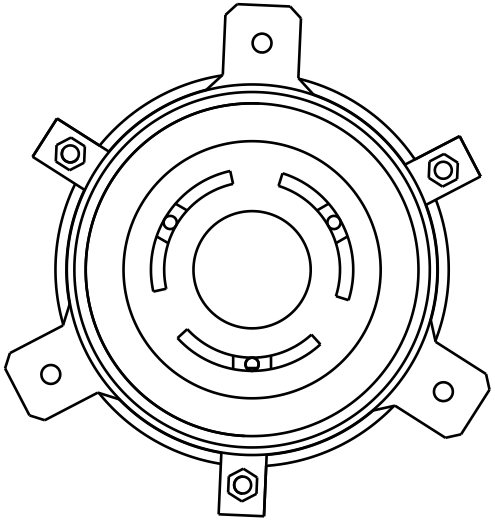
Isometrische Ansicht







Draufsicht



				Werkstoff: PLA	
		Name	Datum	TRINAMIC	
	gezeichnet	Patrick Mattich	12.07.2016	Zeichnungsname: Tischkamera Halterung Zusammenbauzeichnung	
Projektion	Maßstab 1:2			Zeichnungs-Nr.: 8	Index: 01

D. Konfigurationsdateien

Die folgenden Dateien sind Initialisierungsdateien, die zum Betrieb des Testhandlers benötigt werden. Abgebildet sind:

- Initialisierungsdatei der Hardware.
- Initialisierungsdatei für einen TRINAMIC TMC5041 IC.
- Eine Mustervorlage für einen TRINAMIC TMC5041 IC.

Listing D.1: Initialisierungsdatei der Hardware *hardware.ini*

```
[CALIBRATION]
ReferenceSquare=20;

RefCoordinateTopLeftX=14.00
RefCoordinateTopLeftY=385.00
RefCoordinateBottomLeftX=16.00
RefCoordinateBottomLeftY=12.00

PassTrayRefCoordinateBottomLeftX=536.00
PassTrayRefCoordinateBottomLeftY=357.00
PassTrayRefCoordinateBottomRightX=882.00
PassTrayRefCoordinateBottomRightY=357.00

SourceTrayRefCoordinateTopLeftX=536.00
SourceTrayRefCoordinateTopLeftY=204.00
SourceTrayRefCoordinateTopRightX=881.00
SourceTrayRefCoordinateTopRightY=203.00

[DIMENSIONS]
CameraUpXpos=334.0199
CameraUpYpos=84.2297
TestSocketXpos=363.4406
TestSocketYpos=289.5598
CameraDownDeltaX=-32.1
CameraDownDeltaY=83.5

[SOURCETRAY]
vertical=false
referencePoint1X=538.00
referencePoint1Y=200.00
referencePoint2X=883.00
referencePoint2Y=200.00
MechanicalXCorrectionfactor=0.0000
MechanicalYCorrectionfactor=0.0019
RotationForPlacing=0;

[PASSTRAY]
vertical=false
referencePoint1X=538.00
referencePoint1Y=321.00
referencePoint2X=883.00
referencePoint2Y=321.00
MechanicalXCorrectionfactor=-0.0016
MechanicalYCorrectionfactor=-0.001
RotationForPlacing=90

[FAILTRAY]
vertical=true
referencePoint1X=14.00
referencePoint1Y=385.00
referencePoint2X=16.00
referencePoint2Y=12.00
MechanicalXCorrectionfactor=0.0017
MechanicalYCorrectionfactor=0.0010
RotationForPlacing=0;

[XAXIS]
Motor=0
SwitchEndstops=true
SwitchMotorDirection=false
PulleyToothCount=20
```

```
BeltPitchmm=2.0
DisableEndstops=false
VT=800000
A1=1000000
D1=1000000
MaxAcc=1000000
MaxDec=1000000
V1=400000
HomingSpeed=40

[YAXIS]
Motor=2
SwitchEndstops=true
SwitchMotorDirection=false
PulleyToothCount=20
BeltPitchmm=2.0
DisableEndstops=false
VT=800000
A1=1000000
D1=1000000
MaxAcc=1000000
MaxDec=1000000
V1=400000
HomingSpeed=40

[ZAXIS]
Motor=4
SwitchEndstops=false
SwitchMotorDirection=true
StepsPermm=6400
DisableEndstops=false
FullStepResolution=200
VT=102400
A1=1000000
D1=1000000
V1=25600
MaxAcc=1000000
MaxDec=1000000
HomingSpeed=30

[EAXIS]
Motor=5
SwitchMotorDirection=false
Transmission=2.25
Degrees=true
CurrentStandby=60
DisableEndstops=true
VT=25600
A1=1000000
D1=1000000
MaxAcc=1000000
MaxDec=1000000
V1=12800
```

Listing D.2: Initialisierungsdatei für einen TRINAMIC TMC5041

```
[ID]
name=TMC_5041

[TRAYDIMENSIONS]
xFromCorner=11.55
yFromCorner=11.80
xStep=9.40
yStep=9.40
rows=13
columns=32
```

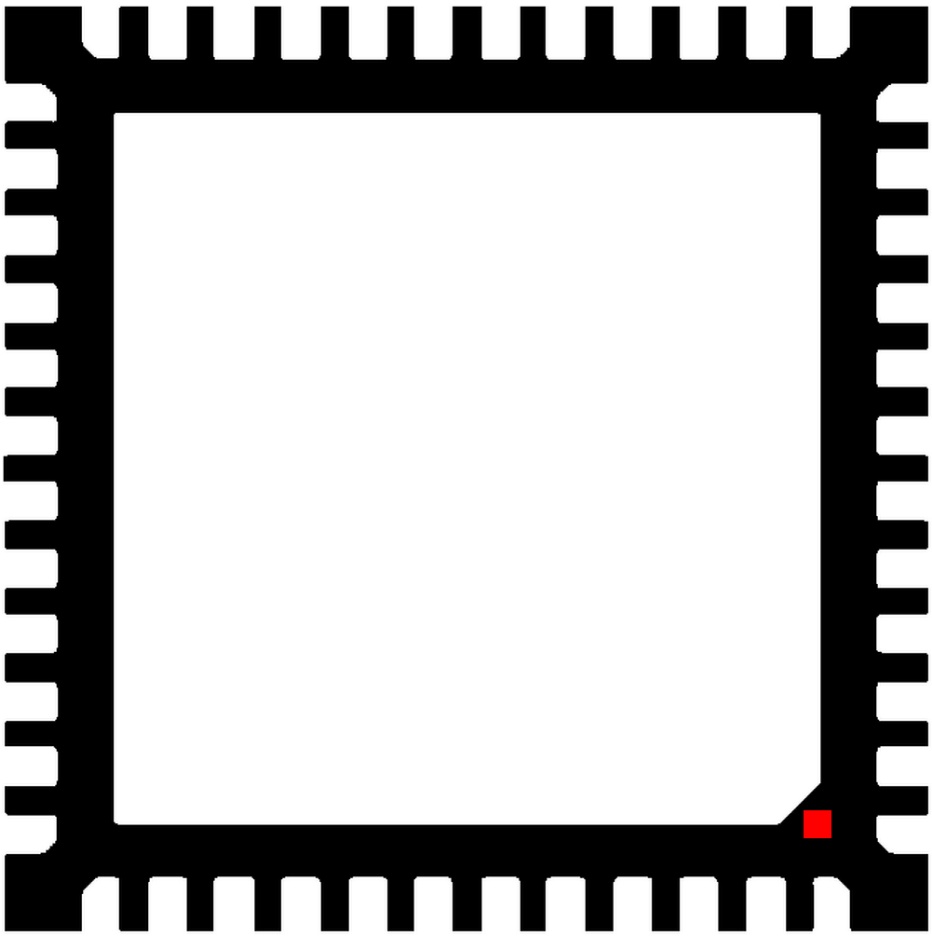


Abbildung D.1.: Mustervorlage für den TMC5041

E. Quellcode der entwickelten Software

Listing E.1: Quellcode von ImageProcessing.h

```
#ifndef IMAGEPROCESSING_H
#define IMAGEPROCESSING_H


#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <QtGui>
#include <stdio.h>
#include <QGraphicsScene>
#include <QMessageBox>

using namespace std;
using namespace cv;

// Winkelversatz bestimmen
double getDegree(Mat &input);

//X-Versatz bestimmen
double getDeltaX(Mat &input);

//Y-Versatz bestimmen
double getDeltaY(Mat &input);

//It returns the biggest contour in an image
RotatedRect getBorders(const Mat&input_img, bool blackAndWhite = false);

//cuts the cropArea out of the input image and returns it
Mat cropImg(const Mat&input, RotatedRect cropArea  );

//rotate input image arround a center point with a given angle
void rotateImg(const Mat&input_img_rot, float angle, Point rot_center);

//draw a filled circle in the middle of an image
void drawCenterPoint( Mat &img );

//draw a crosshair in the middle of an image
void drawCrosshair(Mat &img);

//get the needed pixel for a millimeter
double pixelToMillimeter(Mat &img, double reference);

// cuts out only the area of an image which contains the chip and
// returns the chip image
Mat cropChip(Mat &img);

//This function takes an image an a mastertemplate
//and compares these images in the areas, which are specified by the
//master
float compareImage(Mat &img, Mat &master);

//compare matrix in specified area
float compareMat(const Mat& master, const Mat& image, Rect ROI);

//calculate the degree which the images of the cameras
//need to be rotated
double getCorrectionDegree(Mat &img);

//delete pixels in an image which are
//not the specified color
Mat deleteColor(Mat &img, InputArray color );

//get the pixels in an image which are
//the specified color
Mat extractColor(Mat &img, InputArray color);

//produces an even backround to delete uneven backround which
//could interfere with the image recognition
//the area to cut out needs to be specified
Mat generateEvenBackround(Mat &img, int ROI_X_Startpoint,
                          int ROI_Y_Startpoint, int ROI_width, int ROI_height);


#endif // IMAGEPROCESSING_H
```

Listing E.2: Quellcode von ImageProcessing.cpp

```
#include "ImageProcessing.h"
#include <iostream>
#include <fstream>

#define NORMDIMENSION 1000

using namespace std;
using namespace cv;

// Winkelversatz bestimmen
double getDegree(Mat &input)
{
    RotatedRect boundingBox =  getBorders(input);
    if((boundingBox.angle == 0) && (boundingBox.center == Point2f(0,0)) && (
          boundingBox.size == Size2f(0,0)))
    {
        qDebug() <<Q_FUNC_INFO << "ERROR" ;
        return -1;
    }
    return (/*90 +*/ boundingBox.angle);
}

//X-Versatz bestimmen
double getDeltaX(Mat &input)
{
    RotatedRect boundingBox =  getBorders(input);
    if((boundingBox.angle == 0) && (boundingBox.center == Point2f(0,0)) && (
          boundingBox.size == Size2f(0,0)))
    {
        qDebug() << "ERROR" ;
        return 1;
    }
    Rect boundingBox_straight_temp = getBorders(input).boundingRect();
    rectangle(input,boundingBox_straight_temp,Scalar(0,255,0), 2,8,0);

    //Mittelpunkt vom IC
    circle(input, Point2f(input.cols/2, input.rows/2),4, Scalar(0,0,255),4,8,0);

    //Mittelpunkt von der Kamera (ggf. die Referenz)
    circle(input, boundingBox.center, 4, Scalar(0,255,0),4,8,0);

    double deltaX = input.cols/2 - boundingBox.center.x;
```

```
41      return deltaX;
42  }
43
44  //Y-Versatz bestimmen
45  double getDeltaY(Mat &input)
46  {
47      RotatedRect boundingBox =  getBorders(input);
48      if((boundingBox.angle == 0) && (boundingBox.center == Point2f(0,0)) && (
            boundingBox.size == Size2f(0,0)))
49      {
50          qDebug() << "ERROR" ;
51          return 1;
52      }
53      Rect boundingBox_straight_temp = getBorders(input).boundingRect();
54      rectangle(input,boundingBox_straight_temp,Scalar(0,255,0), 2,8,0);
55
56      //Mittelpunkt vom IC
57      circle(input, Point2f(input.cols/2, input.rows/2),4, Scalar(0,0,255),4,8,0);
58
59      //Mittelpunkt von der Kamera (ggf. die Referenz)
60      circle(input, boundingBox.center, 4, Scalar(0,255,0),4,8,0);
61
62      double deltaY = input.rows/2 - boundingBox.center.y;
63      return deltaY;
64  }
65
66  //It returns the biggest contour in an image
67  // This function is based on an example from the user "Micka":
68  // http://stackoverflow.com/questions/26583649/opencv-c-rectangle-detection-
        which-has-irregular-side (30.06.2016)
69  RotatedRect getBorders(const Mat&input_img, bool blackAndWhite)
70  {
71      RotatedRect boundingBox;
72      Mat mask;
73      Mat gray;
74
75      //check if used image is allready black and white
76      if(blackAndWhite == false)
77      {
78          //convert to grayscale
79          cvtColor(input_img,gray, CV_BGR2GRAY);
80          //binarise image
81          threshold(gray, mask, 0, 255, CV_THRESH_BINARY_INV | CV_THRESH_OTSU);
82      }
83      else
84      {
85          input_img.copyTo(mask);
86      }
87
88      vector<vector<Point>> contours;
89      vector<Vec4i> order;
90
91      //find the outermost contours of each object in the image
92      findContours(mask,contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
93
94      int biggestContourIdx = -1;
95      float biggestContourArea = 0;
96
97      //find the contour, which surrounds the biggest area
98      for( unsigned int i = 0; i< contours.size(); i++ )
99      {
100         float ctArea= contourArea(contours[i]);
101         if(ctArea > biggestContourArea)
102         {
103             biggestContourArea = ctArea;
104             biggestContourIdx = i;
105         }
106     }
107
108     // if no contour was found
109     if(biggestContourIdx < 0)
110     {
111         qDebug() << "No␣contour␣found!" ;
112         return boundingBox;
113     }
114
115     // get the rotatedrect of the largest contour
116     boundingBox = minAreaRect(contours[biggestContourIdx]);
117
118     return boundingBox;
119 }
120
121 //cuts the cropArea out of the input image and returns it
122 Mat cropImg(const Mat&input, RotatedRect cropArea  )
123 {
124     Mat temp;
125     input.copyTo(temp);
126     Rect boundingBox_straight = cropArea.boundingRect();
127     temp = temp(boundingBox_straight);
128     return temp;
129 }
130
131 //rotate input image arround a center point with a given angle
132 void rotateImg(const Mat&input_img_rot, float angle, Point rot_center)
133 {
134     Mat rot_mat = getRotationMatrix2D( rot_center, angle, 1 );
135     warpAffine( input_img_rot, input_img_rot, rot_mat, input_img_rot.size(),
            INTER_LINEAR, BORDER_CONSTANT, Scalar(255,255,255) );
136 }
137
138
139 //draw a filled circle in the middle of an image
140 void drawCenterPoint(Mat &img )
141 {
142     int thickness = -1;//the circle will be drawn filled.
143     int lineType = 8;
144     Point center;
145     center.y = img.rows/2.0;
146     center.x = img.cols/2.0;
147
148     circle( img,
149             center,
150             5.0,
151             Scalar( 255, 0, 0 ),
152             thickness,
153             lineType );
154 }
155
156
157
158
159 //draw a crosshair in the middle of an image
```

```
160 void drawCrosshair(Mat &img)
161 {
162     int thickness = 1;
163     int lineType = 8;
164
165     Point startUp = Point(img.cols/2, 0);
166     Point stopDown = Point(img.cols/2, img.rows);
167     line( img,
168           startUp,
169           stopDown,
170           Scalar( 0, 255, 0 ),
171           thickness,
172           lineType );
173
174     Point startLeft = Point(0, img.rows/2);
175     Point stopRight = Point(img.cols, img.rows/2);
176     line( img,
177           startLeft,
178           stopRight,
179           Scalar( 0, 255, 0 ),
180           thickness,
181           lineType );
182 }
183
184
185 //get the needed pixel for a millimeter
186 double pixelToMillimeter(Mat &img, double reference)
187 {
188     //mirroring should be unnecessary
189     //Camera looks up but the xy coordinates are from the surface -> mirroring
190     Mat image2Temp;
191     img.copyTo(image2Temp);
192     flip(image2Temp, img, 1);//mirroring
193
194     RotatedRect calRect = getBorders(img);
195
196     //check if the reference is a square
197     if(fabs(calRect.size.height - calRect.size.width) > 3)
198     {
199         qDebug()<<"Reference␣not␣a␣square!";
200         return -1.0;
201     }
202     else
203     {
204         qDebug() <<"compFactor:␣" << calRect.size.height / reference;
205         return calRect.size.height / reference;
206
207     }
208     return -1.0;
209 }
210
211
212 // cuts out only the area of an image which contains the chip and
213 // returns the chip image
214 Mat cropChip(Mat &img)
215 {
216     Mat newBackround;
217     //fixed area for the backround
218     generateEvenBackround(img, 160,120,320,240).copyTo(newBackround);
219
220     //get the area of the chip
221     RotatedRect boundingBox =  getBorders(newBackround);
222     if((boundingBox.angle == 0) && (boundingBox.center == Point2f(0,0)) && (
            boundingBox.size == Size2f(0,0)))
223     {
224         cout << "ERROR" << endl;
225         Mat empty;
226         return empty;
227     }
228
229     Mat input_cropped;
230     //cut out the area which contains the chip
231     cropImg(newBackround, boundingBox ).copyTo(input_cropped);
232
233     return input_cropped;
234 }
235
236
237
238 //This function takes an image an a mastertemplate
239 //and compares these images in the areas, which are specified by the
240 //master
241 float compareImage(Mat &img, Mat &master)
242 {
243     if(img.empty() || master.empty())
244         return NULL;
245
246     //delete the read area from the master template
247     Mat masterNoRed = deleteColor(master, Scalar(0,0,255));
248
249     //Resize to NORMDIMENSION
250     Mat input1_norm;
251     Mat input2_norm;
252     resize(masterNoRed, input1_norm, Size(NORMDIMENSION, NORMDIMENSION),0,0,
            INTER_LINEAR);
253     resize(img, input2_norm, Size(NORMDIMENSION, NORMDIMENSION),0,0,INTER_LINEAR
            );
254
255     // convert to grayscale
256     Mat grey1;
257     Mat grey2;
258     cvtColor(input1_norm,grey1, CV_BGR2GRAY);
259     cvtColor(input2_norm,grey2, CV_BGR2GRAY);
260
261     //binarise image
262     Mat master_mask;
263     Mat photo_mask;
264     threshold(grey1, master_mask, 0, 255, CV_THRESH_BINARY_INV | CV_THRESH_OTSU)
            ;
265     threshold(grey2, photo_mask, 0, 255, CV_THRESH_BINARY_INV | CV_THRESH_OTSU);
266
267     //Check if both images were scaled correctly
268     if(!((master_mask.cols == photo_mask.cols) && (master_mask.rows ==
            photo_mask.rows)))
269     {
270         qDebug() << "Matrizen␣muessen␣gleiche␣Dimensionen␣haben!" ;
271         return -1.0;
272     }
273
274     //Get the ROI from the ROI_MASTER_TEMPLATE
275     Mat masterNorm;
```

```
276     resize(master, masterNorm, Size(NORMDIMENSION, NORMDIMENSION),0,0,
            INTER_LINEAR);
277
278     //get the read area of the template, it specifies th area which will be
            compared
279     masterNorm = extractColor(masterNorm, Scalar(0,0,255));
280     RotatedRect rotrect =  getBorders(masterNorm, true);
281     Rect ROI = rotrect.boundingRect();
282
283     // ROI in Input Bild einzeichnen (nur für Darstellung)
284     rectangle(input2_norm,ROI,Scalar(0,255,0), 1,8,0);
285     imshow("SEARCHED AREA", input2_norm);
286
287     float difference;
288     //calculate the difference of pixels in the specified area
289     difference = compareMat(master_mask,photo_mask, ROI);
290
291     return  difference;
292 }
293
294 //compare matrix in specified area
295 float compareMat(const Mat& master,const Mat& image, Rect ROI)
296 {
297     Mat master_mask_ROI;
298     Mat photo_mask_ROI;
299
300     master(ROI).copyTo(master_mask_ROI);
301     image(ROI).copyTo(photo_mask_ROI);
302
303     //Beide Matrizen vergleichen
304     Mat result;
305     result = master_mask_ROI - photo_mask_ROI;
306
307
308     //Weisse (ungleiche) pixel zählen
309     int i,j;
310     int difference = 0;
311
312     for(i=0; i<result.cols; i++)
313     {
314         for(j=0; j<result.rows; j++)
315         {
316             if(master_mask_ROI.at<uchar>(i, j) != photo_mask_ROI.at<uchar>(i, j)
                    )
317             {
318                 difference++;
319             }
320         }
321     }
322     //Auf 0...1 normieren (0 = perfekte Übereinstimmung, 1 = keine
            Übereinstimmung)
323     float similarity_normalized = float(difference) / (result.cols*result.rows);
324     return similarity_normalized;
325 }
326
327 //calculate the degree which the images of the cameras
328 //need to be rotated
329 double getCorrectionDegree(Mat &img)
330 {
331     double angle = getDegree(img);
332     if(fabs(angle) < fabs(angle+90) )
333     {
334         return -angle;
335     }
336     else
337     {
338         return -(angle + 90);
339     }
340 }
341
342 //produces an even backround to delete uneven backround which
343 //could interfere with the image recognition
344 //the area to cut out needs to be specified
345 Mat generateEvenBackround(Mat &img, int ROI_X_Startpoint, int ROI_Y_Startpoint,
        int ROI_width, int ROI_height)
346 {
347     Vec3b intensity = img.at<Vec3b>(ROI_X_Startpoint-1, ROI_Y_Startpoint-1);
348     uchar blue = intensity.val[0];
349     uchar green = intensity.val[1];
350     uchar red = intensity.val[2];
351
352     Mat backround ;
353     img.copyTo(backround);
354     //color new backround
355     backround.setTo(cv::Scalar(blue,green,red));
356
357     //ROI ausschneiden
358     Rect ROI = Rect(ROI_X_Startpoint,ROI_Y_Startpoint,ROI_width,ROI_height);
359     Mat cutOut = img(ROI);
360     cutOut.copyTo(backround(cv::Rect(ROI_X_Startpoint,ROI_Y_Startpoint,cutOut.
            cols, cutOut.rows)));
361
362     return backround;
363 }
364
365 //get the pixels in an image which are
366 //the specified color
367 Mat extractColor(Mat &img, InputArray color)
368 {
369     assert(img.type() == CV_8UC3);
370     Mat specifiedColorOnly;
371     inRange(img, color, color, specifiedColorOnly);
372     return specifiedColorOnly;
373 }
374
375
376 //delete pixels in an image which are
377 //not the specified color
378 Mat deleteColor(Mat &img, InputArray color)
379 {
380     Mat mask, result;
381     img.copyTo(result);
382     inRange(img, Scalar(0,0,255), Scalar(0,0,255), mask);
383     result.setTo(Scalar(0,0,0), mask);
384     return result;
385 }
```

Listing E.3: Quellcode von AxisConfiguration.h

```
1 #ifndef AXISCONFIGURATION_H
```

```
2  #define AXISCONFIGURATION_H
3
4  #include <QtGui>
5  #include <stdio.h>
6  #include "ContainerTypes.h"
7  #include "ModuleInterface.h"
8  #include "TMCL.h"
9  #include <cmath>
10
11 #define X_AXIS 0
12 #define Y_AXIS 1
13 #define Z_AXIS 2
14 #define E_AXIS 3
15
16
17 class AxisConfiguration : public QObject
18 {
19 public:
20     AxisConfiguration();
21     ~AxisConfiguration();
22
23     //Functions
24
25     //Drive to a specified position
26     //Distance in mm
27     //Mode:
28     //true = relative dimensions
29     //false = absolute dimensions
30     void moveToPosition(double distance, bool mode = 0);
31
32     //drive with a specified speed
33     void velocityMode(int speed);
34
35     //stop the motor
36     void motorStop();
37
38     //home the axis
39     void home();
40
41     //set all parameters
42     void init();
43
44     //set an IO port of the module
45     void setIO(int port, int value);
46
47     //get actual position of axis
48     int actualPosition();
49
50     bool homingFinished();
51
52     int actualSpeed();
53
54     bool actualPositionReached();
55
56     int getMaxAcc() const;
57     void setMaxAcc(int value);
58
59     int getVT() const;
60     void setVT(int value);
61
62     int getCurrentRunning() const;
63     void setCurrentRunning(int value);
64
65     int getCurrentStandby() const;
66     void setCurrentStandby(int value);
67
68     int getA1() const;
69     void setA1(int value);
70
71     int getV1() const;
72     void setV1(int value);
73
74     int getMaxDec() const;
75     void setMaxDec(int value);
76
77     int getD1() const;
78     void setD1(int value);
79
80     int getVStart() const;
81     void setVStart(int value);
82
83     int getVStop() const;
84     void setVStop(int value);
85
86     int getMicrostepResolution() const;
87     void setMicrostepResolution(int value);
88
89     bool getDegrees() const;
90     void setDegrees(bool value);
91
92     int getAxis() const;
93     void setAxis(int value);
94
95     QString getName() const;
96     void setName(const QString &value);
97
98     ModuleInterface *getPlugin() const;
99     void setPlugin(ModuleInterface *value);
100
101    bool getSwitchEndstops() const;
102    void setSwitchEndstops(bool value);
103
104    bool getinverseMotor() const;
105    void setinverseMotor(bool value);
106
107    double getTransmission() const;
108    void setTransmission(double value);
109
110    double getBeltPitchmm() const;
111    void setBeltPitchmm(double value);
112
113    int getPulleyToothCount() const;
114    void setPulleyToothCount(int value);
115
116    int getStepsPermm() const;
117    void setStepsPermm(int value);
118
119    int getFullStepResolution() const;
120    void setFullStepResolution(int value);
121
122
123    bool getHomingDirection() const;
```

```
124     void setHomingDirection(bool value);
125
126     bool getDisableEndstops() const;
127     void setDisableEndstops(bool value);
128
129     double getAngle() const;
130
131     int getHomingSpeed() const;
132     void setHomingSpeed(int value);
133
134     void resetActualPosition();
135
136
137 private:
138     //internal parameters
139     int maxAcc ;
140     int VT ;
141     int currentRunning ;
142     int currentStandby ;
143     int A1 ;
144     int V1 ;
145     int maxDec ;
146     int D1 ;
147     int vStart ;
148     int vStop ;
149     int microstepResolution ;
150     int homingSpeed; // mm/sec
151
152     //mechanical parameters
153     int fullStepResolution;
154     double beltPitchmm;
155     int pulleyToothCount;
156     int stepsPermm;
157     bool switchEndstops;
158     int motorDirection;
159     bool inverseMotor;
160     double transmission;
161     bool homingDirection;
162     int homingMode;
163     bool disableEndstops;
164
165     //
166     bool inDegree ;
167     int axis ;
168     QString name;
169     ModuleInterface *plugin;
170     int steps;
171 };
172
173 #endif // AXISCONFIGURATION_H
```

Listing E.4: Quellcode von AxisConfiguration.cpp

```
1  #include "AxisConfiguration.h"
2
3  AxisConfiguration::AxisConfiguration():QObject(0)
4  {
5      qDebug() << Q_FUNC_INFO;
6
7      //Default values
8      maxAcc = 1000000;
9      VT = 500000;
10     currentRunning = 160;
11     currentStandby = 128;
12     A1 = 500000;
13     V1 = 400000;
14     maxDec = 1000000;
15     D1 = 500000;
16     vStart = 1;
17     vStop = 10;
18     microstepResolution = 8;
19     fullStepResolution = 400;//0,9° Resolution | 400 Steps/Rotation
20     beltPitchmm = 2.0 ;//GT2 belt has 2mm pitch
21     pulleyToothCount = 20;
22     inDegree = false;
23     axis = 0;
24     steps = 0;
25     homingSpeed = 50;
26     motorDirection = 1;
27     inverseMotor = true;
28     switchEndstops = true;
29     transmission = 1.0;
30     homingDirection = true;
31     homingMode = 2;
32     disableEndstops = false;
33     stepsPermm = (int)(fullStepResolution * (int)pow(2.0 , (int)this->
           microstepResolution)
34                         * (1/(beltPitchmm * pulleyToothCount)));
35
36 }
37
38 AxisConfiguration::~AxisConfiguration()
39 {
40     this->deleteLater();
41 }
42
43 //Drive to a specified position
44 //Distance in mm
45 //Mode:
46 //true = relative dimensions
47 //false = absolute dimensions
48 void AxisConfiguration::moveToPosition(double distance, bool mode)
49 {
50     qDebug()<<"Distance:␣"<<distance;
51
52     if(!inDegree)
53     {
54         this->steps = 0;
55         this->steps = distance * this->stepsPermm ;
56
57         if(this->plugin)//check if a module is choosen in the comboBox
58         {
59             if(mode)
60             {
61                 Command request(this->plugin->getModuleAddress(), TMCL_MVP, 1,
                       this->axis, motorDirection*this->steps);//REL
62                 Command reply;
63
64                 this->plugin->getCommandHandlerInterface()->doDirectRequest(&
                       request, &reply);
```

```
65              }
66              else
67              {
68                  Command request(this->plugin->getModuleAddress(), TMCL_MVP, 0,
                        this->axis, motorDirection*this->steps);//ABS
69                  Command reply;
70                  this->plugin->getCommandHandlerInterface()->doDirectRequest(&
                        request, &reply);
71              }
72          }
73      }
74      else//Turn a specific angle
75      {
76          qDebug() << __METHOD_NAME__ <<"Degree";
77          this->steps = 0;
78          steps = (int)((distance * fullStepResolution * pow(2.0 , (int)this->
                microstepResolution)) / 360);
79          steps = (int)(steps * transmission);
80
81          if(this->plugin)//check if a module is choosen in the comboBox
82          {
83              if(mode)
84              {
85                  qDebug() <<__METHOD_NAME__<<"relativ";
86                  Command request(this->plugin->getModuleAddress(), TMCL_MVP, 1,
                        this->axis, motorDirection*this->steps);//REL
87                  Command reply;
88                  this->plugin->getCommandHandlerInterface()->doDirectRequest(&
                        request, &reply);
89              }
90              else
91              {
92                  qDebug() <<__METHOD_NAME__<<"abs";
93                  Command request(this->plugin->getModuleAddress(), TMCL_MVP, 0,
                        this->axis, motorDirection*this->steps);//ABS
94                  Command reply;
95                  this->plugin->getCommandHandlerInterface()->doDirectRequest(&
                        request, &reply);
96              }
97          }
98      }
99  }
100
101 //drive with a specified speed
102 void AxisConfiguration::velocityMode(int speed)
103 {   //speed in mm/s
104     int velocity = 0;
105     //from mm/s in microsteps/s
106     velocity = speed * this->stepsPermm;
107
108     if(this->plugin)
109     {
110         Command request(this->plugin->getModuleAddress(), TMCL_ROR, 0, this->
                axis, motorDirection*velocity);
111         Command reply;
112         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
113     }
114 }
115
116 //stop the motor
117 void AxisConfiguration::motorStop()
118 {
119     if (this->plugin)
120     {
121         Command request(this->plugin->getModuleAddress(), TMCL_MST, 0, this->
                axis, 0);
122         Command reply;
123         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
124     }
125 }
126
127 //home the axis
128 void AxisConfiguration::home()
129 {
130     int velocity = 0;
131     //from mm/s in microsteps/s
132     velocity = this->homingSpeed * this->stepsPermm;
133     if (this->plugin)
134     {
135         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 14, this->
                axis, switchEndstops);
136         Command reply;
137         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
138     }
139
140     int add64 = 0;
141     if(!homingDirection)
142         add64 = 64;
143
144     if (this->plugin)
145     {
146         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 193, this->
                axis, homingMode+add64);
147         Command reply;
148         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
149     }
150
151     if (this->plugin)
152     {
153         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 194, this->
                axis, velocity);
154         Command reply;
155         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
156     }
157
158     if (this->plugin)
159     {
160         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 195, this->
                axis, velocity/2);
161         Command reply;
162         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
163     }
164
165     if (this->plugin)
166     {
```

```
167         Command request(this->plugin->getModuleAddress(), TMCL_RFS, 0, this->
                axis, 0);
168         Command reply;
169         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
170     }
171 }
172
173 //set all parameters
174 void AxisConfiguration::init()
175 {
176     if (this->plugin)
177     {
178         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 4, this->
                axis, this->VT);
179         Command reply;
180         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
181     }
182     if (this->plugin)
183     {
184         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 5, this->
                axis, this->maxAcc);
185         Command reply;
186         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
187     }
188     if (this->plugin)
189     {
190         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 6, this->
                axis, this->currentRunning);
191         Command reply;
192         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
193     }
194     if (this->plugin)
195     {
196         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 7, this->
                axis, this->currentStandby);
197         Command reply;
198         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
199     }
200     if (this->plugin)
201     {
202         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 15, this->
                axis, this->A1);
203         Command reply;
204         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
205     }
206     if (this->plugin)
207     {
208         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 16, this->
                axis, this->V1);
209         Command reply;
210         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
211     }
212     if (this->plugin)
213     {
214         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 17, this->
                axis, this->maxDec);
215         Command reply;
216         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
217     }
218     if (this->plugin)
219     {
220         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 18, this->
                axis, this->D1);
221         Command reply;
222         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
223     }
224     if (this->plugin)
225     {
226         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 19, this->
                axis, this->vStart);
227         Command reply;
228         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
229     }
230     if (this->plugin)
231     {
232         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 20, this->
                axis, this->vStop);
233         Command reply;
234         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
235     }
236     if (this->plugin)
237     {
238         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 140, this->
                axis, this->microstepResolution);
239         Command reply;
240         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
241     }
242     if (this->plugin)
243     {
244         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 12, this->
                axis, int(this->disableEndstops));
245         Command reply;
246         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
247     }
248     if (this->plugin)
249     {
250         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 13, this->
                axis, int(this->disableEndstops));
251         Command reply;
252         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
253     }
254     if (this->plugin)
255     {
256         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 127, this->
                axis, 1);
257         Command reply;
258         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
```

```
259     }
260 }
261
262 bool AxisConfiguration::homingFinished()
263 {
264     if (this->plugin)
265     {
266         Command request(this->plugin->getModuleAddress(), TMCL_RFS, 2, this->
                axis, 0);
267         Command reply;
268         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
269
270         if(reply.value != 0)
271             return false;//homing still working
272         else
273             return true;//homing finished
274     }
275     else
276         return true;
277 }
278
279 int AxisConfiguration::actualSpeed()
280 {
281     if (this->plugin)
282     {
283         Command request(this->plugin->getModuleAddress(), TMCL_GAP, 3, this->
                axis, 0);
284         Command reply;
285         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
286
287         return reply.value;
288     }
289     else
290         return 0;
291 }
292
293 //get actual position of axis
294 int AxisConfiguration::actualPosition()
295 {
296     if (this->plugin)
297     {
298         Command request(this->plugin->getModuleAddress(), TMCL_GAP, 1, this->
                axis, 0);
299         Command reply;
300         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
301         if(this->homingDirection)
302             return reply.value;
303         else if(!this->homingDirection)
304             return -reply.value;
305     }
306     else
307         return 0;
308 }
309
310 bool AxisConfiguration::actualPositionReached()
311 {
312     if (this->plugin)
313     {
314         Command request(this->plugin->getModuleAddress(), TMCL_GAP, 8, this->
                axis, 0);
315         Command reply;
316         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
317
318         return (bool)reply.value;
319     }
320     else
321         return 0;
322 }
323
324 //set an IO port of the module
325 void AxisConfiguration::setIO(int port, int value)
326 {
327     if (this->plugin)
328     {
329         Command request(this->plugin->getModuleAddress(), TMCL_SIO, port, 2,
                value);
330         Command reply;
331         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
332     }
333 }
334
335
336 int AxisConfiguration::getMaxAcc() const
337 {
338     return maxAcc;
339 }
340
341 void AxisConfiguration::setMaxAcc(int value)
342 {
343     maxAcc = value;
344 }
345
346 int AxisConfiguration::getVT() const
347 {
348     return VT;
349 }
350
351 void AxisConfiguration::setVT(int value)
352 {
353     VT = value;
354     if (this->plugin)
355     {
356         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 4, this->
                axis, this->VT);
357         Command reply;
358         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
359     }
360 }
361
362 int AxisConfiguration::getCurrentRunning() const
363 {
364     return currentRunning;
365 }
366
367 void AxisConfiguration::setCurrentRunning(int value)
368 {
```

```
369     currentRunning = value;
370 }
371
372 int AxisConfiguration::getCurrentStandby() const
373 {
374     return currentStandby;
375 }
376
377 void AxisConfiguration::setCurrentStandby(int value)
378 {
379     currentStandby = value;
380 }
381
382 int AxisConfiguration::getA1() const
383 {
384     return A1;
385 }
386
387 void AxisConfiguration::setA1(int value)
388 {
389     A1 = value;
390 }
391
392 int AxisConfiguration::getV1() const
393 {
394     return V1;
395 }
396
397 void AxisConfiguration::setV1(int value)
398 {
399     V1 = value;
400 }
401
402 int AxisConfiguration::getMaxDec() const
403 {
404     return maxDec;
405 }
406
407 void AxisConfiguration::setMaxDec(int value)
408 {
409     maxDec = value;
410 }
411
412 int AxisConfiguration::getD1() const
413 {
414     return D1;
415 }
416
417 void AxisConfiguration::setD1(int value)
418 {
419     D1 = value;
420 }
421
422 int AxisConfiguration::getVStart() const
423 {
424     return vStart;
425 }
426
427 void AxisConfiguration::setVStart(int value)
428 {
429     vStart = value;
430 }
431
432 int AxisConfiguration::getVStop() const
433 {
434     return vStop;
435 }
436
437 void AxisConfiguration::setVStop(int value)
438 {
439     vStop = value;
440 }
441
442 int AxisConfiguration::getMicrostepResolution() const
443 {
444     return microstepResolution;
445 }
446
447 void AxisConfiguration::setMicrostepResolution(int value)
448 {
449     microstepResolution = value;
450 }
451
452 int AxisConfiguration::getStepsPermm() const
453 {
454     return stepsPermm;
455 }
456
457 void AxisConfiguration::setStepsPermm(int value)
458 {
459     stepsPermm = value;
460 }
461
462 bool AxisConfiguration::getDegrees() const
463 {
464     return inDegree;
465 }
466
467 void AxisConfiguration::setDegrees(bool value)
468 {
469     inDegree = value;
470 }
471
472 int AxisConfiguration::getAxis() const
473 {
474     return axis;
475 }
476
477 void AxisConfiguration::setAxis(int value)
478 {
479     axis = value;
480 }
481
482 QString AxisConfiguration::getName() const
483 {
484     return name;
485 }
486
487 void AxisConfiguration::setName(const QString &value)
488 {
489     name = value;
490 }
```

```
491
492 ModuleInterface *AxisConfiguration::getPlugin() const
493 {
494     return plugin;
495 }
496
497 void AxisConfiguration::setPlugin(ModuleInterface *value)
498 {
499     plugin = value;
500 }
501
502 bool AxisConfiguration::getSwitchEndstops() const
503 {
504     return switchEndstops;
505 }
506
507 void AxisConfiguration::setSwitchEndstops(bool value)
508 {
509     switchEndstops = value;
510 }
511
512 bool AxisConfiguration::getinverseMotor() const
513 {
514     return inverseMotor;
515 }
516
517 void AxisConfiguration::setinverseMotor(bool value)
518 {
519     inverseMotor = value;
520     if(inverseMotor)
521         motorDirection = -1;
522     else
523         motorDirection = 1;
524 }
525
526 double AxisConfiguration::getTransmission() const
527 {
528     return transmission;
529 }
530
531 void AxisConfiguration::setTransmission(double value)
532 {
533     transmission = value;
534 }
535
536 double AxisConfiguration::getBeltPitchmm() const
537 {
538     return beltPitchmm;
539 }
540
541 void AxisConfiguration::setBeltPitchmm(double value)
542 {
543     beltPitchmm = value;
544 }
545
546 int AxisConfiguration::getPulleyToothCount() const
547 {
548     return pulleyToothCount;
549 }
550
551 void AxisConfiguration::setPulleyToothCount(int value)
552 {
553     pulleyToothCount = value;
554 }
555
556
557 int AxisConfiguration::getFullStepResolution() const
558 {
559     return fullStepResolution;
560 }
561
562 void AxisConfiguration::setFullStepResolution(int value)
563 {
564     fullStepResolution = value;
565 }
566
567 bool AxisConfiguration::getHomingDirection() const
568 {
569     return homingDirection;
570 }
571
572 void AxisConfiguration::setHomingDirection(bool value)
573 {
574     homingDirection = value;
575 }
576
577 bool AxisConfiguration::getDisableEndstops() const
578 {
579     return disableEndstops;
580 }
581
582 void AxisConfiguration::setDisableEndstops(bool value)
583 {
584     disableEndstops = value;
585     if(disableEndstops)
586     {
587         //Right endstop
588         if (this->plugin)
589         {
590             Command request(this->plugin->getModuleAddress(), TMCL_SAP, 12, this
                  ->axis, 1);
591             Command reply;
592             this->plugin->getCommandHandlerInterface()->doDirectRequest(&request
                  , &reply);
593         }
594
595         //Left endstop
596         if (this->plugin)
597         {
598             Command request(this->plugin->getModuleAddress(), TMCL_SAP, 13, this
                  ->axis, 1);
599             Command reply;
600             this->plugin->getCommandHandlerInterface()->doDirectRequest(&request
                  , &reply);
601         }
602     }
603     else
604     {
605         //Right endstop
606         if (this->plugin)
607         {
```

```
608             Command request(this->plugin->getModuleAddress(), TMCL_SAP, 12, this
                    ->axis, 0);
609             Command reply;
610             this->plugin->getCommandHandlerInterface()->doDirectRequest(&request
                    , &reply);
611         }
612
613         //Left endstop
614         if (this->plugin)
615         {
616             Command request(this->plugin->getModuleAddress(), TMCL_SAP, 13, this
                    ->axis, 0);
617             Command reply;
618             this->plugin->getCommandHandlerInterface()->doDirectRequest(&request
                    , &reply);
619         }
620     }
621 }
622
623 double AxisConfiguration::getAngle() const
624 {
625     if (this->plugin)
626     {
627         Command request(this->plugin->getModuleAddress(), TMCL_GAP, 1, this->
                axis, 0);
628         Command reply;
629         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
630
631
632
633         return (360 * (reply.value / (fullStepResolution * pow(2.0 , (int)this->
                microstepResolution)) ) /transmission );
634     }
635     else
636         return 0;
637 }
638
639 int AxisConfiguration::getHomingSpeed() const
640 {
641     return homingSpeed;
642 }
643
644 void AxisConfiguration::setHomingSpeed(int value)
645 {
646     homingSpeed = value;
647 }
648
649 void AxisConfiguration::resetActualPosition()
650 {
651     if (this->plugin)
652     {
653         Command request(this->plugin->getModuleAddress(), TMCL_SAP, 1, this->
                axis, 0);
654         Command reply;
655         this->plugin->getCommandHandlerInterface()->doDirectRequest(&request, &
                reply);
656     }
657 }
```

Listing E.5: Quellcode von TestHandler.h

```
1  /*
2   * TestHandler.h
3   *
4   *  Created on: 17.06.2016
5   *      Author: PM
6   */
7
8  #ifndef TESTHANDLER_H
9  #define TESTHANDLER_H
10
11 //Standard Librarys
12 #include <stdio.h>
13
14 //TRINAMIC Librarys
15 #include "ContainerInterface.h"
16 #include "ModuleInterface.h"
17 #include "ConnectionInterface.h"
18
19 //OPENCV Library
20 #include <opencv2/opencv.hpp>
21 #include <opencv2/highgui/highgui.hpp>
22
23 //Qt Librarys
24 #include <QtGui>
25 #include <QMap>
26 #include <QGraphicsScene>
27 #include <QList>
28 #include <QFileDialog>
29 #include <QMessageBox>
30 #include <QFile>
31 #include <QSettings>
32 #include <QDir>
33 #include <QFile>
34
35 //My Files
36 #include "ImageProcessing.h"
37 #include "TestSocketDummy.h"
38 #include "AxisConfiguration.h"
39 #include "Tray.h"
40
41
42 using namespace std;
43 using namespace cv;
44
45 namespace Ui
46 {
47 class TestHandler;
48 }
49
50 class TestHandler : public ContainerInterface
51 {
52     Q_INTERFACES(ContainerInterface)
53     Q_OBJECT
54     Q_PLUGIN_METADATA(IID ContainerInterface_idd FILE "../Container.json")
55
56 public:
57     TestHandler();
58     TestHandler(QString key, PluginHandler *PluginHandler);
59     ContainerInterface* clone(QString key, PluginHandler *pluginHandler);
```

```
60      ~TestHandler();
61      void init(QWidget *parent);
62      QString getVersion();
63
64  private:
65      //variables
66      bool videoStreamOnOff = false;
67      QString moduleKey;
68      int axis;
69      Ui::TestHandler *ui;
70      PluginHandler* pluginHandler;
71      Mat image1, image2;
72      VideoCapture source1, source2;
73      QGraphicsScene *scene1,*scene2, *sceneTemplate;
74      AxisConfiguration XAxis;
75      AxisConfiguration YAxis;
76      AxisConfiguration ZAxis;
77      AxisConfiguration EAxis;
78      int timerIDcamera;
79      int timerIDpolling;
80      bool motorStopped;
81      bool vacPipetteEmpty;
82
83      //Dimensions and Trays
84      Tray sourceTray;
85      Tray passTray;
86      Tray failTray;
87      Mat chipTemplate;
88      double testSocketXpos;
89      double testSocketYpos;
90      double CameraDownDeltaX;
91      double CameraDownDeltaY;
92      void goToFailTray();
93      void goToPassTray();
94      void goToSourceTray();
95
96      //Calibration
97      int referenceSize;
98      double RefCoordinateTopLeftX;
99      double RefCoordinateTopLeftY;
100     double RefCoordinateBottomLeftX;
101     double RefCoordinateBottomLeftY;
102     double correctionFactorY;
103     double cam1CorDeg;
104     double cam2CorDeg;
105     double pixelTommFactorCam1;
106     double pixelTommFactorCam2;
107     double dX, dY;
108     void measureYCompansation();
109     bool goToCenterCam1();
110     bool goToCenterCam2();
111     void adjustRotation();
112     bool adjustChip();
113     void calAllTrays();
114
115     //FSM
116     int state;
117     bool firstChip;
118     double xZero, yZero;
119     bool correctPosition;
120     void testFSM();
121     bool stopFSM;
122     bool testResult;
123
124     //Camera
125     void startCamera();
126     QImage mat_to_qimage_ref(Mat &mat, QImage::Format format);
127     void goToCamera();
128     double cameraUpXpos;
129     double cameraUpYpos;
130
131     //functions
132     void initComboBoxes();
133     void waitForMotorStop();
134     void moveToXY(double x, double y);
135     void goToTestSocket();
136     void startTestSocket();
137
138     //LoadSaveSettings
139     QSettings *hardwareSettings;
140     QSettings *chipSettings;
141     bool checkIfKeyExists(QSettings *settings, QString keyName, QString
            groupName);
142     void initHardwareSettings(QString filename);
143     void initChipSettings(QString filename);
144     QString getDatapathBrowser(QString filetype);
145     void loadHardwareSettings();
146     void writeHardwareSettings();
147     void loadChipSettings();
148     void writeChipSettings();
149     void loadChipTemplate();
150     Mat input;
151
152     //GUI
153     void updateModuleSettingsGUI();
154
155     //TestSocket
156     TestSocketDummy testSocket;
157
158 protected:
159     void timerEvent(QTimerEvent*ev);
160
161 private slots:
162     void finished(int code = -1);
163     void updateModuleList();
164     void saveModuleList();
165
166     //GUI
167     void on_calCam2pushButton_clicked();
168     void on_calCam1pushButton_clicked();
169     void on_stopTestpushButton_clicked();
170     void on_placeICpushButton_clicked();
171     void on_loadSettingspushButton_clicked();
172     void on_sendRelativeButton_clicked();
173     void on_sendAbsoluteButton_clicked();
174     void on_turnCam1PluspushButton_clicked();
175     void on_turnCam1MinuspushButton_clicked();
176     void on_turnCam2PluspushButton_clicked();
177     void on_turnCam2MinuspushButton_clicked();
178     void on_centerCam1pushButton_clicked();
179     void on_calibrateYpushButton_clicked();
180     void on_cropImagepushButton_clicked();
```

```
181     void on_homeXButton_clicked();
182     void on_homeYButton_clicked();
183     void on_homeZButton_clicked();
184     void on_getDegreepushButton_clicked();
185     void on_getICpushButton_clicked();
186     void on_updateModulesButton_clicked();
187     void on_saveModuleListButton_clicked();
188     void on_homeButton_clicked();
189     void on_upButton_pressed();
190     void on_upButton_released();
191     void on_downButton_pressed();
192     void on_downButton_released();
193     void on_leftButton_pressed();
194     void on_leftButton_released();
195     void on_rightButton_pressed();
196     void on_rightButton_released();
197     void on_ZupButton_pressed();
198     void on_ZupButton_released();
199     void on_ZdownButton_pressed();
200     void on_ZdownButton_released();
201     void on_loadChipSettingspushButton_clicked();
202     void on_loadChipTemplatepushButton_clicked();
203     void on_gotoCamerapushButton_clicked();
204     void on_gotoTestSocketpushButton_clicked();
205     void on_failTraypushButton_clicked();
206     void on_sourceTraypushButton_clicked();
207     void on_passTraypushButton_clicked();
208     void on_centerCam2pushButton_clicked();
209     void on_centerChippushButton_clicked();
210     void on_calPasspushButton_clicked();
211     void on_calSourcepushButton_clicked();
212     void on_SourcegoToChippushButton_clicked();
213     void on_SourcetoZeropushButton_clicked();
214     void on_PassgoToChippushButton_clicked();
215     void on_PasstoZeropushButton_clicked();
216     void on_FailgoToChippushButton_clicked();
217     void on_FailtoZeropushButton_clicked();
218     void on_calalltrayspushButton_clicked();
219     void on_startAutomatedTestpushButton_clicked();
220     void on_stopAutomatedTestpushButton_clicked();
221     void on_resetCounterpushButton_clicked();
222     void changeResultLabel(int result);
223     void errorMessage(QString errorCode);
224 };
225
226 #endif // TESTHANDLER_H
```

Listing E.6: Quellcode von TestHandler.cpp

```
1  /*
2   * TestHandler.cpp
3   *
4   *  Created on: 17.05.2016
5   *      Author: PM
6   */
7
8  #include "TestHandler.h"
9  #include "ContainerTypes.h"
10 #include "ui_TestHandler.h"
11 #include <opencv2/opencv.hpp>
12 #include <QtGui>
13 #include <stdio.h>
14 #include <QGraphicsTextItem>
15 #include "ImageProcessing.h"
16 #include "TestSocketDummy.h"
17 #include <iostream>
18 #include <fstream>
19
20 using namespace std;
21 using namespace cv;
22
23 #define MODULE_NAME "TestHandler"
24 #define MODULE_ICON ":/Misc/Wizard"
25
26 Q_PLUGIN_METADATA(IID ContainerInterface_idd)
27 Q_EXPORT_PLUGIN2(container_test_handler, TestHandler)
28
29 /*************************************************
30  *
31  * Constructor and Deconstructor
32  *
33  *************************************************/
34 TestHandler::TestHandler() : ContainerInterface(QString("undefined"),
35                                                 QString(MODULE_NAME),
                                                        ContainerType::TestHandler
                                                        , NULL, MODULE_ICON), ui
                                                        (0)
36 {
37     qDebug() << __METHOD_NAME__;
38 }
39
40 //Constructor
41 TestHandler::TestHandler(QString key, PluginHandler *pluginHandler) :
42     ContainerInterface(key, QString(MODULE_NAME), ContainerType::TestHandler,
43                        pluginHandler, MODULE_ICON),ui(new Ui::TestHandler)
44 {
45     qDebug() << __METHOD_NAME__;
46     ui->setupUi(this);
47     //is needed for overlaying the container and the plugins
48     setProperty(DynamicProperty::WindowFlags, Qt::SubWindow);
49     setProperty(DynamicProperty::WindowModal, false);
50     setProperty(DynamicProperty::WindowAddSizeGrip, true);
51
52     //initiliaze the axis pluginHandler with a blank
53     XAxis.setPlugin(getPluginHandler()->getModulePlugin(QString("")));
54     YAxis.setPlugin(getPluginHandler()->getModulePlugin(QString("")));
55     ZAxis.setPlugin(getPluginHandler()->getModulePlugin(QString("")));
56     EAxis.setPlugin(getPluginHandler()->getModulePlugin(QString("")));
57
58     sourceTray.setXAxis(&XAxis);
59     sourceTray.setYAxis(&YAxis);
60     sourceTray.setZAxis(&ZAxis);
61     sourceTray.setEAxis(&EAxis);
62
63     passTray.setXAxis(&XAxis);
64     passTray.setYAxis(&YAxis);
65     passTray.setZAxis(&ZAxis);
66     passTray.setEAxis(&EAxis);
67
68     failTray.setXAxis(&XAxis);
```

```
69      failTray.setYAxis(&YAxis);
70      failTray.setZAxis(&ZAxis);
71      failTray.setEAxis(&EAxis);
72
73      timerIDcamera = 0;
74      timerIDpolling = 0;
75      cam1CorDeg = 0.0;
76      cam2CorDeg = 0.0;
77      pixelTommFactorCam1 = 10.0;
78      pixelTommFactorCam2 = 16.0;
79      ui->loadSettingspushButton->setEnabled(false);
80      ui->tab->setEnabled(false);
81
82      QObject::connect(&testSocket,SIGNAL(finished(int)),this, SLOT(
            changeResultLabel(int)));
83      QObject::connect(&testSocket, SIGNAL(error(QString)),this, SLOT(errorMessage
            (QString)));
84      QObject::connect(&testSocket, SIGNAL(state(int)),ui->progressBar,SLOT(
            setValue(int)) );
85      QObject::connect(&testSocket, SIGNAL(stateMax(int)),ui->progressBar,SLOT(
            setMaximum(int)));
86
87      ui->progressBar->setMinimum(0);
88      ui->progressBar->setMaximum(1);
89      ui->progressBar->setValue(0);
90  }
91
92  TestHandler::~TestHandler()
93  {
94      qDebug() << __METHOD_NAME__;
95      if (ui)
96          delete ui;
97  }
98
99  /*****************************************************
100  *
101  * Module connections
102  *
103  *****************************************************/
104
105 void TestHandler::updateModuleList()
106 {
107     QStringList moduleKeys = getPluginHandler()->getModulePluginKeys(false);
108
109     ui->XModulecomboBox->clear();
110     ui->YModulecomboBox->clear();
111     ui->ZModulecomboBox->clear();
112     ui->EModulecomboBox->clear();
113     ui->XAxiscomboBox->clear();
114     ui->YAxiscomboBox->clear();
115     ui->ZAxiscomboBox->clear();
116     ui->EAxiscomboBox->clear();
117
118     foreach(QString moduleKey, moduleKeys)
119     {
120         if (ModuleInterface *plugin = getPluginHandler()->getModulePlugin(
                moduleKey))
121         {
122             ui->XModulecomboBox->addItem(QString(moduleKey));
123             ui->YModulecomboBox->addItem(QString(moduleKey));
124             ui->ZModulecomboBox->addItem(QString(moduleKey));
125             ui->EModulecomboBox->addItem(QString(moduleKey));
126
127             for (int i = 0; i < plugin->getNumberOfAxes(); i++)
128             {
129                 ui->XAxiscomboBox->addItem(QString::number(i));
130                 ui->YAxiscomboBox->addItem(QString::number(i));
131                 ui->ZAxiscomboBox->addItem(QString::number(i));
132                 ui->EAxiscomboBox->addItem(QString::number(i));
133             }
134         }
135     }
136 }
137
138
139 void TestHandler::saveModuleList()
140 {
141
142
143     qDebug() << __METHOD_NAME__;
144     XAxis.setPlugin(getPluginHandler()->getModulePlugin(ui->XModulecomboBox->
            currentText()));
145     XAxis.setAxis(ui->XAxiscomboBox->currentText().toInt());
146
147     YAxis.setPlugin(getPluginHandler()->getModulePlugin(ui->YModulecomboBox->
            currentText()));
148     YAxis.setAxis(ui->YAxiscomboBox->currentText().toInt());
149
150     ZAxis.setPlugin(getPluginHandler()->getModulePlugin(ui->ZModulecomboBox->
            currentText()));
151     ZAxis.setAxis(ui->ZAxiscomboBox->currentText().toInt());
152
153     EAxis.setPlugin(getPluginHandler()->getModulePlugin(ui->EModulecomboBox->
            currentText()));
154     EAxis.setAxis(ui->EAxiscomboBox->currentText().toInt());
155
156     //Its only possible to initialize the axis when the module plugin
157     // is initialized
158
159     //Only allow to init the axis AFTER the user chose a module for each axis
160     if((ui->XModulecomboBox->currentText() != "<none>") &&
161             (ui->YModulecomboBox->currentText() != "<none>") &&
162             (ui->ZModulecomboBox->currentText() != "<none>") &&
163             (ui->EModulecomboBox->currentText() != "<none>"))
164     {
165         ui->loadSettingspushButton->setEnabled(true);
166
167     }
168
169     if(ui->XSwitchEndstopscomboBox->currentText()== "No")
170         XAxis.setSwitchEndstops(false);
171     else
172         XAxis.setSwitchEndstops(true);
173
174     if(ui->YSwitchEndstopscomboBox->currentText()== "No")
175         YAxis.setSwitchEndstops(false);
176     else
177         YAxis.setSwitchEndstops(true);
178
179     if(ui->ZSwitchEndstopscomboBox->currentText()== "No")
180         ZAxis.setSwitchEndstops(false);
181     else
```

```
182         ZAxis.setSwitchEndstops(true);
183
184
185     if(ui->XMotorDirectioncomboBox->currentText() == "No")
186     {
187         XAxis.setinverseMotor(false);
188         qDebug() << __METHOD_NAME__ <<"XMotorDirectioncomboBox:␣"<<ui->
                XMotorDirectioncomboBox->currentText();
189     }
190     else
191     {
192         XAxis.setinverseMotor(true);
193     }
194
195     if(ui->YMotorDirectioncomboBox->currentText() == "No")
196     {
197         YAxis.setinverseMotor(false);
198         qDebug() << __METHOD_NAME__ <<"YMotorDirectioncomboBox:␣"<<ui->
                YMotorDirectioncomboBox->currentText();
199     }
200     else
201     {
202         YAxis.setinverseMotor(true);
203     }
204
205     if(ui->ZMotorDirectioncomboBox->currentText() == "No")
206     {
207         ZAxis.setinverseMotor(false);
208         qDebug() << __METHOD_NAME__ <<"ZMotorDirectioncomboBox:␣"<<ui->
                ZMotorDirectioncomboBox->currentText();
209     }
210     else
211     {
212         ZAxis.setinverseMotor(true);
213     }
214
215     if(ui->XhomingDirectioncomboBox->currentText() == "Left")
216     {
217         XAxis.setHomingDirection(true);
218     }
219     else
220     {
221         XAxis.setHomingDirection(false);
222     }
223
224     if(ui->YhomingDirectioncomboBox->currentText() == "Left")
225     {
226         YAxis.setHomingDirection(true);
227     }
228     else
229     {
230         YAxis.setHomingDirection(false);
231     }
232
233     if(ui->ZhomingDirectioncomboBox->currentText() == "Left")
234     {
235         ZAxis.setHomingDirection(true);
236     }
237     else
238     {
239         ZAxis.setHomingDirection(false);
240     }
241
242 }
243
244 /*************************************************
245  *
246  * Container functions
247  *
248  *************************************************/
249 ContainerInterface* TestHandler::clone(QString key, PluginHandler *pluginHandler
        )
250 {
251     qDebug() << __METHOD_NAME__;
252     return new TestHandler(key, pluginHandler);
253 }
254
255 void TestHandler::init(QWidget *parent)//is called every time the testhandler
       container is opened,
256 //not only onne time like the constructor
257 {
258     qDebug() << __METHOD_NAME__;
259     if(parent != parentWidget())
260         qWarning() <<Q_FUNC_INFO<<parent << parentWidget();
261
262     QObject::connect(static_cast<QDialog*>(parentWidget()), SIGNAL(finished(int)
           ), this, SLOT(finished(int)));
263
264
265
266     ui->XModulecomboBox->resetLastIndex();
267     ui->YModulecomboBox->resetLastIndex();
268     ui->ZModulecomboBox->resetLastIndex();
269     ui->EModulecomboBox->resetLastIndex();
270
271     updateModuleList();
272     initComboBoxes();
273
274     this->startCamera();
275 }
276
277 void TestHandler::finished(int code)//is called every time the testhandler
       container is closed,
278 //not only onne time like the destructor
279 {
280     qDebug() << __METHOD_NAME__;
281     Q_UNUSED(code);
282     parentWidget()->deleteLater();
283     setParent(0);
284
285     if(timerIDcamera != 0)
286     {
287         killTimer(timerIDcamera);
288         timerIDcamera = 0;
289     }
290
291     if(timerIDpolling != 0)
292     {
293         killTimer(timerIDpolling);
294         timerIDpolling = 0;
295     }
296
```

```
297     if(source1.isOpened())  // check if we succeeded
298         source1.release();
299
300     if(source2.isOpened())  // check if we succeeded
301         source2.release();
302 }
303
304 QString TestHandler::getVersion()
305 {
306     qDebug() << __METHOD_NAME__;
307     return QString(APP_VERSION_STRING);
308 }
309
310 /*************************************************
311  *
312  * Timer
313  *
314  *************************************************/
315 void TestHandler::timerEvent(QTimerEvent* ev)
316 {
317     if (ev->timerId() == timerIDcamera)
318     {
319
320
321         ui->xActualPoslabel->setText(QString::number((double)XAxis.
                actualPosition()/ XAxis.getStepsPermm(),'f', 4));
322         ui->yActualPoslabel->setText(QString::number((double)YAxis.
                actualPosition()/ YAxis.getStepsPermm(),'f', 4));
323         ui->zActualPoslabel->setText(QString::number((double)ZAxis.
                actualPosition()/ ZAxis.getStepsPermm(),'f', 4));
324         ui->eActualPoslabel->setText(QString::number(EAxis.getAngle(),'f', 4));
325
326         QImage temp;
327
328         //The clear() is important to prevent scenes to add up and clog the
                memory
329         scene1->clear();
330         scene1->setBackgroundBrush(QColor(0, 0, 0)); // Schwarzer Hintergrund
331         source1.operator >>(image1);
332         if(image1.empty())
333         {
334             //Debug message no good idea, because it would be displayed with the
335             //framerate speed and would spam the log
336             return;
337         }
338         else
339         {
340             resize(image1, image1,Size(300,225));//4:3 Format
341
342             //because of historical reasons opencv uses the BGR format while in
                    Qt the morecommon RGB
343             //format is used, so for the real colors it has to be converted
344             cvtColor(image1,image1,COLOR_BGR2RGB);
345             rotateImg(image1, cam1CorDeg, Point(image1.cols/2.0, image1.rows
                    /2.0));
346
347             drawCrosshair(image1);
348
349             temp = mat_to_qimage_ref(image1, QImage::Format_RGB888 );
350             scene1->addPixmap(QPixmap::fromImage(temp));
351             ui->ViewBox1->setScene((scene1));
352         }
353
354         //The clear() is important to prevent scenes to add up and clog the
                memory
355         scene2->clear();
356         scene2->setBackgroundBrush(QColor(0, 0, 0)); // Schwarzer Hintergrund
357         source2.operator >>(image2);
358         if(image2.empty())
359         {
360             return;
361         }
362         else
363         {
364             //do the resize first, so that the following instructions
365             //dont have to process too much data
366             resize(image2, image2,Size(300,225));//4:3 Format
367
368             //because of historical reasons opencv uses the BGR format while in
                    Qt the morecommon RGB
369             //format is used, so for the real colors it has to be converted
370             cvtColor(image2,image2,COLOR_BGR2RGB);
371
372             Mat image2Temp;
373             image2.copyTo(image2Temp);
374             flip(image2Temp, image2, 1);
375
376             rotateImg(image2, cam2CorDeg, Point(image2.cols/2.0, image2.rows
                    /2.0));
377             drawCrosshair(image2);
378             //drawCenterPoint(image2);
379             temp = mat_to_qimage_ref(image2, QImage::Format_RGB888 );
380             scene2->addPixmap(QPixmap::fromImage(temp));
381             ui->ViewBox2->setScene((scene2));
382         }
383     }
384     else if(ev->timerId() == timerIDpolling)
385     {
386         if(XAxis.actualSpeed()!=0 || YAxis.actualSpeed()!=0 || ZAxis.actualSpeed
                ()!=0 || EAxis.actualSpeed()!=0)
387         {
388             //motors running
389             motorStopped = 0;
390         }
391         else
392         {
393             //motors stopped
394             motorStopped = 1;
395         }
396     }
397 }
398
399 //wait untill all motors stopped
400 //the whiles loops are in an extra thread,
401 //so the program is not blocked
402 void TestHandler::waitForMotorStop()
403 {
404     while(YAxis.actualSpeed() != 0 )
405     {
406         QApplication::processEvents();
407     }
408     while(XAxis.actualSpeed() != 0 )
```

```
409     {
410         QApplication::processEvents();
411     }
412     while(ZAxis.actualSpeed() != 0 )
413     {
414         QApplication::processEvents();
415     }
416     while(EAxis.actualSpeed() != 0 )
417     {
418         QApplication::processEvents();
419     }
420 }
421
422 //if the y axis is moved its necessary to compensate the not 90° angle with the
        x axis.
423 void TestHandler::moveToXY(double x, double y)
424 {
425     double compensationDistance = y * correctionFactorY;
426
427     this->XAxis.moveToPosition((x + compensationDistance), true);
428     this->YAxis.moveToPosition(y, true);
429
430     waitForMotorStop();
431 }
432
433 //Move to the testsocket Position
434 void TestHandler::goToTestSocket()
435 {
436     XAxis.moveToPosition(this->testSocketXpos, false);
437     YAxis.moveToPosition(this->testSocketYpos, false);
438     waitForMotorStop();
439 }
440
441 //Start the testsocket
442 void TestHandler::startTestSocket()
443 {
444     testSocket.setName(sourceTray.getName());
445     ui->resultLabel->setStyleSheet("");
446     ui->resultLabel->setText("Wait...");
447     testSocket.start();
448 }
449
450 //Go to the reference markers of the Failtray
451 //and set the measured positions  in the Failtray Object
452 void TestHandler::goToFailTray()
453 {
454     double deltaX, deltaY, xTL, yTL, xBL, yBL;
455
456     XAxis.moveToPosition(this->failTray.getReferencePoint2X(), false);
457     YAxis.moveToPosition(this->failTray.getReferencePoint2Y(), false);
458
459     waitForMotorStop();
460
461     this->on_centerCam1pushButton_clicked();
462
463     xBL = ui->xActualPoslabel->text().toDouble();
464     yBL = ui->yActualPoslabel->text().toDouble();
465
466     //############################################
467
468     XAxis.moveToPosition(this->failTray.getReferencePoint1X(), false);
469     YAxis.moveToPosition(this->failTray.getReferencePoint1Y(), false);
470
471     waitForMotorStop();
472
473     this->on_centerCam1pushButton_clicked();
474
475     xTL = ui->xActualPoslabel->text().toDouble();
476     yTL = ui->yActualPoslabel->text().toDouble();
477
478     //experimentell ermittelte werte
479     moveToXY((30 - 0.4 +0.1 ), (-30 + 0.4 -0.4 +0.4));
480
481     deltaX = xTL - xBL;
482     deltaY = yTL - yBL;
483
484     failTray.setXZero(ui->xActualPoslabel->text().toDouble());
485     failTray.setYZero(ui->yActualPoslabel->text().toDouble());
486
487 }
488
489 //Go to the reference markers of the Passtray
490 //and set the measured positions  in the Psstray Object
491 void TestHandler::goToPassTray()
492 {
493     double deltaX, deltaY, xTL, yTL, xBL, yBL;
494
495     XAxis.moveToPosition(this->passTray.getReferencePoint2X(), false);
496     YAxis.moveToPosition(this->passTray.getReferencePoint2Y(), false);
497
498     waitForMotorStop();
499
500     this->on_centerCam1pushButton_clicked();
501
502     xBL = ui->xActualPoslabel->text().toDouble();
503     yBL = ui->yActualPoslabel->text().toDouble();
504
505     //############################################
506
507     XAxis.moveToPosition(this->passTray.getReferencePoint1X(), false);
508     YAxis.moveToPosition(this->passTray.getReferencePoint1Y(), false);
509
510     waitForMotorStop();
511
512     this->on_centerCam1pushButton_clicked();
513
514     xTL = ui->xActualPoslabel->text().toDouble();
515     yTL = ui->yActualPoslabel->text().toDouble();
516
517     //experimentell ermittelte werte
518     moveToXY(14.9, 15);
519
520     deltaX = xTL - xBL;
521     deltaY = yTL - yBL;
522
523     passTray.setXZero(ui->xActualPoslabel->text().toDouble());
524     passTray.setYZero(ui->yActualPoslabel->text().toDouble());
525 }
526
527 //Go to the reference markers of the Sourcetray
528 //and set the measured positions  in the Sourcetray Object
529 void TestHandler::goToSourceTray()
```

```
{
    double deltaX, deltaY, xTL, yTL, xBL, yBL;

    XAxis.moveToPosition(this->sourceTray.getReferencePoint2X(), false);
    YAxis.moveToPosition(this->sourceTray.getReferencePoint2Y(), false);

    waitForMotorStop();

    this->on_centerCam1pushButton_clicked();

    xBL = ui->xActualPoslabel->text().toDouble();
    yBL = ui->yActualPoslabel->text().toDouble();

    //##############################################

    XAxis.moveToPosition(this->sourceTray.getReferencePoint1X(), false);
    YAxis.moveToPosition(this->sourceTray.getReferencePoint1Y(), false);

    waitForMotorStop();

    this->on_centerCam1pushButton_clicked();

    xTL = ui->xActualPoslabel->text().toDouble();
    yTL = ui->yActualPoslabel->text().toDouble();

    //experimentell ermittelte werte
    moveToXY((15 - 0.84 + 0.1), (-(15+135.9) + 0.28 - 0.5 + 0.2));
    deltaX = xTL - xBL;
    deltaY = yTL - yBL;

    sourceTray.setXZero(ui->xActualPoslabel->text().toDouble());
    sourceTray.setYZero(ui->yActualPoslabel->text().toDouble());
}
```

Listing E.7: Quellcode von LoadSaveSettings.cpp

```
#include "TestHandler.h"
#include <QtGui>
#include "ui_TestHandler.h"
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>

//checks if a given QSettings (the .ini file) contains specific key
//in a specific group
bool TestHandler::checkIfKeyExists(QSettings *settings, QString keyName, QString
      groupName)
{
    bool status = false;
    //safe the current group for returning to it later
    QString tempGroup = settings->group();

    settings->endGroup();
    if(keyName!=NULL && groupName!=NULL)
    {
        settings->beginGroup(groupName);
        status = settings->contains(keyName);
        settings->endGroup();
        qDebug()<< keyName <<status;
        settings->beginGroup(tempGroup);
        return status;
    }
    else
    {
        return false;
    }
    return false;
}

//Initialize the hardware settings variable
void TestHandler::initHardwareSettings(QString filename)
{
    hardwareSettings = new QSettings(filename,QSettings::IniFormat);
}

void TestHandler::initChipSettings(QString filename)
{
    chipSettings = new QSettings(filename,QSettings::IniFormat);
}

//Opens a file browser an let the user choose a file to get it's
//filepath which is then returned
QString TestHandler::getDatapathBrowser(QString filetype)
{
    //Parameter for FileDialog
    QString filename = QFileDialog::getOpenFileName(
                this,
                "Open File",
                "C://",
                //"Setting Files (*.ini);;Text Files (*.txt);;All Files (*.*)"
                filetype
                );

    //Pop up Message Box which shows the choosen filepath
    QMessageBox::information(this,"Data Path", "Your chosen file is : "+filename
         );

    return filename;
}

//Loads the hardware ini file and configures the notion control module
void TestHandler::loadHardwareSettings()
{
    //after a setting file is loaded the machine can be controlled
    ui->tab->setEnabled(true);

    //initialize qsettings
    initHardwareSettings(getDatapathBrowser("Setting Files (*.ini)"));

    //write the specific parameters from the .ini in the axis object variables
    writeHardwareSettings();

    //initialize each axis with a default parameters so it can move
    XAxis.init();
    YAxis.init();
    ZAxis.init();
    EAxis.init();

    updateModuleSettingsGUI();
}

```

```
83  //Write the parameters from the .ini file into the TMC Hardware
84  //and program variables
85  void TestHandler::writeHardwareSettings()
86  {
87      //Calibration
88      hardwareSettings->beginGroup("CALIBRATION");
89
90      if(checkIfKeyExists(hardwareSettings, "ReferenceSquare", "CALIBRATION" ))
91          this->referenceSize = hardwareSettings->value("ReferenceSquare").toInt()
                ;
92
93      if(checkIfKeyExists(hardwareSettings, "RefCoordinateTopLeftX", "CALIBRATION"
            ))
94          this->RefCoordinateTopLeftX = hardwareSettings->value("
                RefCoordinateTopLeftX").toDouble();
95
96      if(checkIfKeyExists(hardwareSettings, "RefCoordinateTopLeftY", "CALIBRATION"
            ))
97          this->RefCoordinateTopLeftY = hardwareSettings->value("
                RefCoordinateTopLeftY").toDouble();
98
99      if(checkIfKeyExists(hardwareSettings, "RefCoordinateBottomLeftX", "
            CALIBRATION" ))
100         this->RefCoordinateBottomLeftX = hardwareSettings->value("
                RefCoordinateBottomLeftX").toDouble();
101
102     if(checkIfKeyExists(hardwareSettings, "RefCoordinateBottomLeftY", "
            CALIBRATION" ))
103         this->RefCoordinateBottomLeftY = hardwareSettings->value("
                RefCoordinateBottomLeftY").toDouble();
104
105     hardwareSettings->endGroup();
106
107
108     //Dimensions
109     hardwareSettings->beginGroup("DIMENSIONS");
110
111     if(checkIfKeyExists(hardwareSettings, "CameraUpXpos", "DIMENSIONS" ))
112         this->cameraUpXpos = hardwareSettings->value("CameraUpXpos").toDouble();
113
114     if(checkIfKeyExists(hardwareSettings, "CameraUpYpos", "DIMENSIONS" ))
115         this->cameraUpYpos = hardwareSettings->value("CameraUpYpos").toDouble();
116
117     if(checkIfKeyExists(hardwareSettings, "TestSocketXpos", "DIMENSIONS" ))
118         this->testSocketXpos = hardwareSettings->value("TestSocketXpos").
                toDouble();
119
120     if(checkIfKeyExists(hardwareSettings, "TestSocketYpos", "DIMENSIONS" ))
121         this->testSocketYpos = hardwareSettings->value("TestSocketYpos").
                toDouble();
122
123     if(checkIfKeyExists(hardwareSettings, "CameraDownDeltaX", "DIMENSIONS" ))
124     {
125         this->CameraDownDeltaX = hardwareSettings->value("CameraDownDeltaX").
                toDouble();
126         this->passTray.setCameraDownDeltaX(hardwareSettings->value("
                CameraDownDeltaX").toDouble());
127         this->failTray.setCameraDownDeltaX(hardwareSettings->value("
                CameraDownDeltaX").toDouble());
128         this->sourceTray.setCameraDownDeltaX(hardwareSettings->value("
                CameraDownDeltaX").toDouble());
129     }
130
131     if(checkIfKeyExists(hardwareSettings, "CameraDownDeltaY", "DIMENSIONS" ))
132     {
133         this->CameraDownDeltaY = hardwareSettings->value("CameraDownDeltaY").
                toDouble();
134         this->passTray.setCameraDownDeltaY(hardwareSettings->value("
                CameraDownDeltaY").toDouble());
135         this->failTray.setCameraDownDeltaY(hardwareSettings->value("
                CameraDownDeltaY").toDouble());
136         this->sourceTray.setCameraDownDeltaY(hardwareSettings->value("
                CameraDownDeltaY").toDouble());
137     }
138
139     hardwareSettings->endGroup();
140
141
142     //These parameters must not change with different ICs,
143     //they describe the orientation of the Trays, which is mechanically
144     //fixed on the table
145     hardwareSettings->beginGroup("SOURCETRAY");
146     if(checkIfKeyExists(hardwareSettings, "vertical", "SOURCETRAY" ))
147     {
148         this->sourceTray.setVertical(hardwareSettings->value("vertical").toBool
                ());
149     }
150
151     if(checkIfKeyExists(hardwareSettings, "referencePoint1X", "SOURCETRAY" ))
152     {
153         this->sourceTray.setReferencePoint1X(hardwareSettings->value("
                referencePoint1X").toDouble());
154     }
155
156     if(checkIfKeyExists(hardwareSettings, "referencePoint1Y", "SOURCETRAY" ))
157     {
158         this->sourceTray.setReferencePoint1Y(hardwareSettings->value("
                referencePoint1Y").toDouble());
159     }
160
161     if(checkIfKeyExists(hardwareSettings, "referencePoint2X", "SOURCETRAY" ))
162     {
163         this->sourceTray.setReferencePoint2X(hardwareSettings->value("
                referencePoint2X").toDouble());
164     }
165
166     if(checkIfKeyExists(hardwareSettings, "referencePoint2Y", "SOURCETRAY" ))
167     {
168         this->sourceTray.setReferencePoint2Y(hardwareSettings->value("
                referencePoint2Y").toDouble());
169     }
170
171     if(checkIfKeyExists(hardwareSettings, "MechanicalXCorrectionfactor", "
            SOURCETRAY" ))
172     {
173         this->sourceTray.setMechanicalXCorrectionfactor(hardwareSettings->value(
                "MechanicalXCorrectionfactor").toDouble());
174     }
175
176     if(checkIfKeyExists(hardwareSettings, "MechanicalYCorrectionfactor", "
            SOURCETRAY" ))
177     {
```

```
178         this->sourceTray.setMechanicalYCorrectionfactor(hardwareSettings->value(
                "MechanicalYCorrectionfactor").toDouble());
179     }
180
181     if(checkIfKeyExists(hardwareSettings, "RotationForPlacing", "SOURCETRAY" ))
182     {
183         this->sourceTray.setRotationForPlacing(hardwareSettings->value("
                RotationForPlacing").toDouble());
184     }
185     hardwareSettings->endGroup();
186
187
188     hardwareSettings->beginGroup("PASSTRAY");
189     if(checkIfKeyExists(hardwareSettings, "vertical", "PASSTRAY" ))
190     {
191         this->passTray.setVertical(hardwareSettings->value("vertical").toBool())
                ;
192     }
193
194     if(checkIfKeyExists(hardwareSettings, "referencePoint1X", "PASSTRAY" ))
195     {
196         this->passTray.setReferencePoint1X(hardwareSettings->value("
                referencePoint1X").toDouble());
197     }
198
199     if(checkIfKeyExists(hardwareSettings, "referencePoint1Y", "PASSTRAY" ))
200     {
201         this->passTray.setReferencePoint1Y(hardwareSettings->value("
                referencePoint1Y").toDouble());
202     }
203
204     if(checkIfKeyExists(hardwareSettings, "referencePoint2X", "PASSTRAY" ))
205     {
206         this->passTray.setReferencePoint2X(hardwareSettings->value("
                referencePoint2X").toDouble());
207     }
208
209     if(checkIfKeyExists(hardwareSettings, "referencePoint2Y", "PASSTRAY" ))
210     {
211         this->passTray.setReferencePoint2Y(hardwareSettings->value("
                referencePoint2Y").toDouble());
212     }
213
214     if(checkIfKeyExists(hardwareSettings, "MechanicalXCorrectionfactor", "
            PASSTRAY" ))
215     {
216         this->passTray.setMechanicalXCorrectionfactor(hardwareSettings->value("
                MechanicalXCorrectionfactor").toDouble());
217     }
218
219     if(checkIfKeyExists(hardwareSettings, "MechanicalYCorrectionfactor", "
            PASSTRAY" ))
220     {
221         this->passTray.setMechanicalYCorrectionfactor(hardwareSettings->value("
                MechanicalYCorrectionfactor").toDouble());
222     }
223
224     if(checkIfKeyExists(hardwareSettings, "RotationForPlacing", "PASSTRAY" ))
225     {
226         this->passTray.setRotationForPlacing(hardwareSettings->value("
                RotationForPlacing").toDouble());
227     }
228     hardwareSettings->endGroup();
229
230
231     hardwareSettings->beginGroup("FAILTRAY");
232     if(checkIfKeyExists(hardwareSettings, "vertical", "FAILTRAY" ))
233     {
234         this->failTray.setVertical(hardwareSettings->value("vertical").toBool())
                ;
235     }
236
237     if(checkIfKeyExists(hardwareSettings, "referencePoint1X", "FAILTRAY" ))
238     {
239         this->failTray.setReferencePoint1X(hardwareSettings->value("
                referencePoint1X").toDouble());
240     }
241
242     if(checkIfKeyExists(hardwareSettings, "referencePoint1Y", "FAILTRAY" ))
243     {
244         this->failTray.setReferencePoint1Y(hardwareSettings->value("
                referencePoint1Y").toDouble());
245     }
246
247     if(checkIfKeyExists(hardwareSettings, "referencePoint2X", "FAILTRAY" ))
248     {
249         this->failTray.setReferencePoint2X(hardwareSettings->value("
                referencePoint2X").toDouble());
250     }
251
252     if(checkIfKeyExists(hardwareSettings, "referencePoint2Y", "FAILTRAY" ))
253     {
254         this->failTray.setReferencePoint2Y(hardwareSettings->value("
                referencePoint2Y").toDouble());
255     }
256
257     if(checkIfKeyExists(hardwareSettings, "MechanicalXCorrectionfactor", "
            FAILTRAY" ))
258     {
259         this->failTray.setMechanicalXCorrectionfactor(hardwareSettings->value("
                MechanicalXCorrectionfactor").toDouble());
260     }
261
262     if(checkIfKeyExists(hardwareSettings, "MechanicalYCorrectionfactor", "
            FAILTRAY" ))
263     {
264         this->failTray.setMechanicalYCorrectionfactor(hardwareSettings->value("
                MechanicalYCorrectionfactor").toDouble());
265     }
266
267     if(checkIfKeyExists(hardwareSettings, "RotationForPlacing", "FAILTRAY" ))
268     {
269         this->failTray.setRotationForPlacing(hardwareSettings->value("
                RotationForPlacing").toDouble());
270     }
271     hardwareSettings->endGroup();
272
273
274     //X Axis
275     hardwareSettings->beginGroup("XAXIS");
276
277     if(checkIfKeyExists(hardwareSettings, "Motor", "XAXIS" ))
```

```
278         XAxis.setAxis(hardwareSettings->value("Motor").toInt());
279
280     if(checkIfKeyExists(hardwareSettings, "SwitchEndstops", "XAXIS" ))
281         XAxis.setSwitchEndstops(hardwareSettings->value("SwitchEndstops").toBool
                ());
282
283     if(checkIfKeyExists(hardwareSettings, "SwitchMotorDirection", "XAXIS" ))
284         XAxis.setinverseMotor(hardwareSettings->value("SwitchMotorDirection").
                toBool());
285
286     if(checkIfKeyExists(hardwareSettings, "PulleyToothCount", "XAXIS" ))
287         XAxis.setPulleyToothCount(hardwareSettings->value("PulleyToothCount").
                toInt());
288
289     if(checkIfKeyExists(hardwareSettings, "BeltPitchmm", "XAXIS" ))
290         XAxis.setBeltPitchmm(hardwareSettings->value("BeltPitchmm").toFloat());
291
292     if(checkIfKeyExists(hardwareSettings, "DisableEndstops", "XAXIS" ))
293         XAxis.setDisableEndstops(hardwareSettings->value("DisableEndstops").
                toBool());
294
295     if(checkIfKeyExists(hardwareSettings, "VT", "XAXIS" ))
296         XAxis.setVT(hardwareSettings->value("VT").toInt());
297
298     if(checkIfKeyExists(hardwareSettings, "A1", "XAXIS" ))
299         XAxis.setA1(hardwareSettings->value("A1").toInt());
300
301     if(checkIfKeyExists(hardwareSettings, "D1", "XAXIS" ))
302         XAxis.setD1(hardwareSettings->value("D1").toInt());
303
304     if(checkIfKeyExists(hardwareSettings, "V1", "XAXIS" ))
305         XAxis.setV1(hardwareSettings->value("V1").toInt());
306
307     if(checkIfKeyExists(hardwareSettings, "MaxAcc", "XAXIS" ))
308         XAxis.setMaxAcc(hardwareSettings->value("MaxAcc").toInt());
309
310     if(checkIfKeyExists(hardwareSettings, "MaxDec", "XAXIS" ))
311         XAxis.setMaxDec(hardwareSettings->value("MaxDec").toInt());
312
313     if(checkIfKeyExists(hardwareSettings, "HomingSpeed", "XAXIS" ))
314         XAxis.setHomingSpeed(hardwareSettings->value("HomingSpeed").toInt());
315
316     hardwareSettings->endGroup();
317
318
319     //Y Axis
320     hardwareSettings->beginGroup("YAXIS");
321
322     if(checkIfKeyExists(hardwareSettings, "Motor", "YAXIS" ))
323         YAxis.setAxis(hardwareSettings->value("Motor").toInt());
324
325     if(checkIfKeyExists(hardwareSettings, "SwitchEndstops", "YAXIS" ))
326         YAxis.setSwitchEndstops(hardwareSettings->value("SwitchEndstops").toBool
                ());
327
328     if(checkIfKeyExists(hardwareSettings, "SwitchMotorDirection", "YAXIS" ))
329         YAxis.setinverseMotor(hardwareSettings->value("SwitchMotorDirection").
                toBool());
330
331     if(checkIfKeyExists(hardwareSettings, "PulleyToothCount", "YAXIS" ))
332         YAxis.setPulleyToothCount(hardwareSettings->value("PulleyToothCount").
                toInt());
333
334     if(checkIfKeyExists(hardwareSettings, "BeltPitchmm", "YAXIS" ))
335         YAxis.setBeltPitchmm(hardwareSettings->value("BeltPitchmm").toFloat());
336
337     if(checkIfKeyExists(hardwareSettings, "DisableEndstops", "YAXIS" ))
338         YAxis.setDisableEndstops(hardwareSettings->value("DisableEndstops").
                toBool());
339
340     if(checkIfKeyExists(hardwareSettings, "VT", "YAXIS" ))
341         YAxis.setVT(hardwareSettings->value("VT").toInt());
342
343     if(checkIfKeyExists(hardwareSettings, "A1", "YAXIS" ))
344         YAxis.setA1(hardwareSettings->value("A1").toInt());
345
346     if(checkIfKeyExists(hardwareSettings, "D1", "YAXIS" ))
347         YAxis.setD1(hardwareSettings->value("D1").toInt());
348
349     if(checkIfKeyExists(hardwareSettings, "V1", "YAXIS" ))
350         YAxis.setV1(hardwareSettings->value("V1").toInt());
351
352     if(checkIfKeyExists(hardwareSettings, "MaxAcc", "YAXIS" ))
353         YAxis.setMaxAcc(hardwareSettings->value("MaxAcc").toInt());
354
355     if(checkIfKeyExists(hardwareSettings, "MaxDec", "YAXIS" ))
356         YAxis.setMaxDec(hardwareSettings->value("MaxDec").toInt());
357
358     if(checkIfKeyExists(hardwareSettings, "HomingSpeed", "YAXIS" ))
359         YAxis.setHomingSpeed(hardwareSettings->value("HomingSpeed").toInt());
360 //      YAxis.setAngleCompensation(-0.00505738183);
361
362
363     hardwareSettings->endGroup();
364
365     //Z Axis
366     hardwareSettings->beginGroup("ZAXIS");
367     if(checkIfKeyExists(hardwareSettings, "Motor", "ZAXIS" ))
368         ZAxis.setAxis(hardwareSettings->value("Motor").toInt());
369
370     if(checkIfKeyExists(hardwareSettings, "SwitchEndstops", "ZAXIS" ))
371         ZAxis.setSwitchEndstops(hardwareSettings->value("SwitchEndstops").toBool
                ());
372
373     if(checkIfKeyExists(hardwareSettings, "SwitchMotorDirection", "ZAXIS" ))
374         ZAxis.setinverseMotor(hardwareSettings->value("SwitchMotorDirection").
                toBool());
375
376     if(checkIfKeyExists(hardwareSettings, "StepsPermm", "ZAXIS" ))
377         ZAxis.setStepsPermm(hardwareSettings->value("StepsPermm").toInt());
378
379     if(checkIfKeyExists(hardwareSettings, "FullStepResolution", "ZAXIS" ))
380         ZAxis.setFullStepResolution(hardwareSettings->value("FullStepResolution"
                ).toInt());
381
382     if(checkIfKeyExists(hardwareSettings, "VT", "ZAXIS" ))
383         ZAxis.setVT(hardwareSettings->value("VT").toInt());
384
385     if(checkIfKeyExists(hardwareSettings, "A1", "ZAXIS" ))
386         ZAxis.setA1(hardwareSettings->value("A1").toInt());
387
```

```
388     if(checkIfKeyExists(hardwareSettings, "D1", "ZAXIS" ))
389         ZAxis.setD1(hardwareSettings->value("D1").toInt());
390
391     if(checkIfKeyExists(hardwareSettings, "V1", "ZAXIS" ))
392         ZAxis.setV1(hardwareSettings->value("V1").toInt());
393
394     if(checkIfKeyExists(hardwareSettings, "MaxAcc", "ZAXIS" ))
395         ZAxis.setMaxAcc(hardwareSettings->value("MaxAcc").toInt());
396
397     if(checkIfKeyExists(hardwareSettings, "MaxDec", "ZAXIS" ))
398         ZAxis.setMaxDec(hardwareSettings->value("MaxDec").toInt());
399
400     if(checkIfKeyExists(hardwareSettings, "HomingSpeed", "ZAXIS" ))
401         ZAxis.setHomingSpeed(hardwareSettings->value("HomingSpeed").toInt());
402
403     if(checkIfKeyExists(hardwareSettings, "DisableEndstops", "ZAXIS" ))
404         ZAxis.setDisableEndstops(hardwareSettings->value("DisableEndstops").
                toBool());
405
406
407     hardwareSettings->endGroup();
408
409     //E Axis
410     hardwareSettings->beginGroup("EAXIS");
411     if(checkIfKeyExists(hardwareSettings, "Motor", "EAXIS" ))
412         EAxis.setAxis(hardwareSettings->value("Motor").toInt());
413
414     if(checkIfKeyExists(hardwareSettings, "SwitchMotorDirection", "EAXIS" ))
415         EAxis.setinverseMotor(hardwareSettings->value("SwitchMotorDirection").
                toBool());
416
417     if(checkIfKeyExists(hardwareSettings, "Transmission", "EAXIS" ))
418         EAxis.setTransmission(hardwareSettings->value("Transmission").toFloat())
                ;
419
420     if(checkIfKeyExists(hardwareSettings, "Degrees", "EAXIS" ))
421         EAxis.setDegrees(hardwareSettings->value("Degrees").toBool());
422
423     if(checkIfKeyExists(hardwareSettings, "CurrentStandby", "EAXIS" ))
424         EAxis.setCurrentStandby(hardwareSettings->value("CurrentStandby").toInt
                ());
425
426     if(checkIfKeyExists(hardwareSettings, "DisableEndstops", "EAXIS" ))
427         EAxis.setDisableEndstops(hardwareSettings->value("DisableEndstops").
                toBool());
428
429     if(checkIfKeyExists(hardwareSettings, "VT", "EAXIS" ))
430         EAxis.setVT(hardwareSettings->value("VT").toInt());
431
432     if(checkIfKeyExists(hardwareSettings, "A1", "EAXIS" ))
433         EAxis.setA1(hardwareSettings->value("A1").toInt());
434
435     if(checkIfKeyExists(hardwareSettings, "D1", "EAXIS" ))
436         EAxis.setD1(hardwareSettings->value("D1").toInt());
437
438     if(checkIfKeyExists(hardwareSettings, "V1", "EAXIS" ))
439         EAxis.setV1(hardwareSettings->value("V1").toInt());
440
441     if(checkIfKeyExists(hardwareSettings, "MaxAcc", "EAXIS" ))
442         EAxis.setMaxAcc(hardwareSettings->value("MaxAcc").toInt());
443
444     if(checkIfKeyExists(hardwareSettings, "MaxDec", "EAXIS" ))
445         EAxis.setMaxDec(hardwareSettings->value("MaxDec").toInt());
446
447
448     hardwareSettings->endGroup();
449 }
450
451 //load the chip setting specified for each chipmodell
452 //and writes the variables
453 void TestHandler::loadChipSettings()
454 {
455     //initialize qsettings
456     initChipSettings(getDatapathBrowser("Setting␣Files␣(*.ini)"));
457
458     //write the specific parameters from the .ini in the axis object variables
459     writeChipSettings();
460 }
461
462 //write the parameters from the loaded chip ini file
463 //into the variables
464 void TestHandler::writeChipSettings()
465 {
466     //The following parameters change with different chips
467     chipSettings->beginGroup("ID");
468     if(checkIfKeyExists(chipSettings, "name", "ID" ))
469     {
470         this->failTray.setName(chipSettings->value("name").toString());
471         this->passTray.setName(chipSettings->value("name").toString());
472         this->sourceTray.setName(chipSettings->value("name").toString());
473     }
474     chipSettings->endGroup();
475
476
477     chipSettings->beginGroup("TRAYDIMENSIONS");
478     if(checkIfKeyExists(chipSettings, "xFromCorner", "TRAYDIMENSIONS" ))
479     {
480         this->failTray.setXFromCorner(chipSettings->value("xFromCorner").
                toDouble());
481         this->passTray.setXFromCorner(chipSettings->value("xFromCorner").
                toDouble());
482         this->sourceTray.setXFromCorner(chipSettings->value("xFromCorner").
                toDouble());
483     }
484
485     if(checkIfKeyExists(chipSettings, "yFromCorner", "TRAYDIMENSIONS" ))
486     {
487         this->failTray.setYFromCorner(chipSettings->value("yFromCorner").
                toDouble());
488         this->passTray.setYFromCorner(chipSettings->value("yFromCorner").
                toDouble());
489         this->sourceTray.setYFromCorner(chipSettings->value("yFromCorner").
                toDouble());
490     }
491
492     if(checkIfKeyExists(chipSettings, "xStep", "TRAYDIMENSIONS" ))
493     {
494         this->failTray.setXStep(chipSettings->value("xStep").toDouble());
495         this->passTray.setXStep(chipSettings->value("xStep").toDouble());
496         this->sourceTray.setXStep(chipSettings->value("xStep").toDouble());
497     }
498
```

```
499     if(checkIfKeyExists(chipSettings, "yStep", "TRAYDIMENSIONS" ))
500     {
501         this->failTray.setYStep(chipSettings->value("yStep").toDouble());
502         this->passTray.setYStep(chipSettings->value("yStep").toDouble());
503         this->sourceTray.setYStep(chipSettings->value("yStep").toDouble());
504     }
505
506     if(checkIfKeyExists(chipSettings, "rows", "TRAYDIMENSIONS" ))
507     {
508         this->failTray.setRows(chipSettings->value("rows").toInt());
509         this->passTray.setRows(chipSettings->value("rows").toInt());
510         this->sourceTray.setRows(chipSettings->value("rows").toInt());
511     }
512
513     if(checkIfKeyExists(chipSettings, "columns", "TRAYDIMENSIONS" ))
514     {
515         this->failTray.setColumns(chipSettings->value("columns").toInt());
516         this->passTray.setColumns(chipSettings->value("columns").toInt());
517         this->sourceTray.setColumns(chipSettings->value("columns").toInt());
518     }
519
520     chipSettings->endGroup();
521 }
522
523 //load master template
524 void TestHandler::loadChipTemplate()
525 {
526     QString imagePath = getDatapathBrowser("Pictures␣(*.png)");
527
528     //opencv need a different filepath format then the one QT delivers
529     imagePath.replace("/","\\\\");
530
531     Mat input = cv::imread(imagePath.toStdString());
532     if( !input.data )
533     {
534         qDebug() << "␣No␣picture␣found!";
535         return ;
536     }
537
538     input.copyTo(this->chipTemplate);
539
540     sceneTemplate = new QGraphicsScene();
541     //The clear() is important to prevent scenes to add up and clog the memory/*
542     sceneTemplate->clear();
543     sceneTemplate->setBackgroundBrush(QColor(0, 0, 0)); // Schwarzer Hintergrund
544     cv::resize(input, input,Size(300,225));//4:3 Format
545     cvtColor(input,input,COLOR_BGR2RGB);
546     QImage temp = mat_to_qimage_ref(input, QImage::Format_RGB888 );
547     sceneTemplate->addPixmap(QPixmap::fromImage(temp));
548     ui->masterTemplateViewBox->setScene((sceneTemplate));
549
550 }
```

Listing E.8: Quellcode von GUI.cpp

```
1  #include "TestHandler.h"
2  #include <QtGui>
3  #include "ui_TestHandler.h"
4  #include <chrono>
5  #include <thread>
6  //#define DEBUG_ON
7  /*****************************************************
8   *
9   *
10  *  Tab 1 buttons and other control input
11  *
12  *
13 /*****************************************************/
14 void TestHandler::on_homeButton_clicked()
15 {
16 }
17
18 void TestHandler::on_upButton_pressed()
19 {
20     YAxis.velocityMode(80);
21 }
22
23 void TestHandler::on_upButton_released()
24 {
25     YAxis.motorStop();
26 }
27
28 void TestHandler::on_downButton_pressed()
29 {
30     YAxis.velocityMode(-80);
31 }
32
33 void TestHandler::on_downButton_released()
34 {
35     YAxis.motorStop();
36 }
37
38 void TestHandler::on_rightButton_pressed()
39 {
40     XAxis.velocityMode(80);
41 }
42
43 void TestHandler::on_rightButton_released()
44 {
45     XAxis.motorStop();
46 }
47
48 void TestHandler::on_leftButton_pressed()
49 {
50     XAxis.velocityMode(-80);
51 }
52
53 void TestHandler::on_leftButton_released()
54 {
55     XAxis.motorStop();
56 }
57
58 void TestHandler::on_ZupButton_pressed()
59 {
60     ZAxis.velocityMode(80);
61 }
62
63 void TestHandler::on_ZupButton_released()
64 {
65     ZAxis.motorStop();
```

```
66  }
67
68  void TestHandler::on_ZdownButton_pressed()
69  {
70      ZAxis.velocityMode(-80);
71  }
72
73  void TestHandler::on_ZdownButton_released()
74  {
75      ZAxis.motorStop();
76  }
77
78  void TestHandler::on_homeXButton_clicked()
79  {
80      XAxis.home();
81  }
82
83  void TestHandler::on_homeYButton_clicked()
84  {
85      YAxis.home();
86  }
87
88  void TestHandler::on_homeZButton_clicked()
89  {
90      ZAxis.home();
91  }
92
93  void TestHandler::on_sendRelativeButton_clicked()
94  {
95      double compensationDistance = ui->YManualValuelineEdit->text().toDouble() *
            correctionFactorY;
96      XAxis.moveToPosition((ui->XManualValuelineEdit->text().toDouble() +
            compensationDistance), true);
97      YAxis.moveToPosition(ui->YManualValuelineEdit->text().toDouble(), true);
98      ZAxis.moveToPosition(ui->ZManualValuelineEdit->text().toDouble(), true);
99      EAxis.moveToPosition(ui->EManualValuelineEdit->text().toDouble(), true);
100 }
101
102 void TestHandler::on_sendAbsoluteButton_clicked()
103 {
104     XAxis.moveToPosition(ui->XManualValuelineEdit->text().toDouble(), false);
105     YAxis.moveToPosition(ui->YManualValuelineEdit->text().toDouble(), false);
106 }
107
108 void TestHandler::on_turnCam1PluspushButton_clicked()
109 {
110     cam1CorDeg = cam1CorDeg + 90;
111 }
112
113 void TestHandler::on_turnCam1MinuspushButton_clicked()
114 {
115     cam1CorDeg = cam1CorDeg - 90;
116 }
117
118 void TestHandler::on_turnCam2PluspushButton_clicked()
119 {
120     cam2CorDeg = cam2CorDeg + 90;
121 }
122
123 void TestHandler::on_turnCam2MinuspushButton_clicked()
124 {
125     cam2CorDeg = cam2CorDeg - 90;
126
127 }
128
129 void TestHandler::on_getDegreepushButton_clicked()
130 {
131     adjustRotation();
132 }
133
134 void TestHandler::on_getICpushButton_clicked()
135 {
136     XAxis.moveToPosition(CameraDownDeltaX, true);
137     YAxis.moveToPosition(CameraDownDeltaY,true);
138
139     waitForMotorStop();
140
141     ZAxis.velocityMode(-10);
142     std::this_thread::sleep_for(std::chrono::milliseconds(50));
143     while(ZAxis.actualSpeed() != 0)
144     {
145         QApplication::processEvents();
146     }
147
148     XAxis.setIO(0,0);//Vacuum on
149
150     std::this_thread::sleep_for(std::chrono::milliseconds(200));
151
152     ZAxis.moveToPosition(-5.0, false);
153     std::this_thread::sleep_for(std::chrono::milliseconds(50));
154     while(ZAxis.actualSpeed() != 0)
155     {
156         QApplication::processEvents();
157     }
158
159     XAxis.moveToPosition(-CameraDownDeltaX, true);
160     YAxis.moveToPosition(-CameraDownDeltaY,true);
161     waitForMotorStop();
162 }
163
164
165 void TestHandler::on_calCam1pushButton_clicked()
166 {
167     waitForMotorStop();
168
169     Mat freshImage1;
170     source1.operator >> (freshImage1);
171
172     cam1CorDeg = getDegree(freshImage1) ;
173
174     rotateImg(freshImage1, cam1CorDeg, Point(freshImage1.cols/2.0, freshImage1.
            rows/2.0));
175     pixelTommFactorCam1 = pixelToMillimeter(freshImage1, referenceSize);
176 }
177
178
179 void TestHandler::on_calCam2pushButton_clicked()
180 {
181     waitForMotorStop();
182
183     Mat freshImage1;
184     source2.operator >> (freshImage1);
```

```
185
186     Mat image2Temp;
187     freshImage1.copyTo(image2Temp);
188     flip(image2Temp, freshImage1, 1);//flip image because cam2 looks up from
            under the table
189
190     cam2CorDeg = getDegree(freshImage1) ;
191
192     rotateImg(freshImage1, cam2CorDeg, Point(freshImage1.cols/2.0, freshImage1.
            rows/2.0));
193     pixelTommFactorCam2 = pixelToMillimeter(freshImage1, referenceSize);
194 }
195
196 void TestHandler::on_stopTestpushButton_clicked()
197 {
198     killTimer(timerIDpolling);
199     stopFSM = true;
200 }
201
202 void TestHandler::on_placeICpushButton_clicked()
203 {
204     XAxis.moveToPosition(CameraDownDeltaX, true);
205     YAxis.moveToPosition(CameraDownDeltaY,true);
206     waitForMotorStop();
207
208     XAxis.setIO(0,0);
209
210     ZAxis.velocityMode(-10);
211     while(ZAxis.actualSpeed() != 0)
212     {
213         QApplication::processEvents();
214     }
215     std::this_thread::sleep_for(std::chrono::milliseconds(200));
216     XAxis.setIO(0,1);
217     std::this_thread::sleep_for(std::chrono::milliseconds(1000));
218
219     int vt = ZAxis.getVT();
220     ZAxis.setVT(3000);
221     ZAxis.moveToPosition(1, true);
222     waitForMotorStop();
223
224     XAxis.moveToPosition(-2, true);
225     waitForMotorStop();
226     ZAxis.setVT(vt);
227
228     ZAxis.moveToPosition(0, false);
229
230     while(ZAxis.actualSpeed() != 0)
231     {
232         QApplication::processEvents();
233     }
234
235     XAxis.moveToPosition(2, true);
236     //drive the zaxis out of the switching zone
237     ZAxis.moveToPosition(-5.0, false);
238     while(ZAxis.actualSpeed() != 0)
239     {
240         QApplication::processEvents();
241     }
242
243     XAxis.moveToPosition(-CameraDownDeltaX, true);
244     YAxis.moveToPosition(-CameraDownDeltaY,true);
245 }
246
247 void TestHandler::on_centerCam2pushButton_clicked()
248 {
249     bool check = false;
250     while(!check)
251     {
252         check = goToCenterCam2();
253         waitForMotorStop();
254     }
255 }
256
257 void TestHandler::on_centerCam1pushButton_clicked()
258 {
259     bool check = false;
260     while(!check)
261     {
262         check = goToCenterCam1();
263         waitForMotorStop();
264     }
265 }
266
267 void TestHandler::on_calibrateYpushButton_clicked()
268 {
269     this->measureYCompansation();
270 }
271
272
273
274 void TestHandler::on_calSourcepushButton_clicked()
275 {
276     double deltaX, deltaY, xR, yR, xL, yL;
277
278     XAxis.moveToPosition(sourceTray.getReferencePoint1X(), false);
279     YAxis.moveToPosition(sourceTray.getReferencePoint1Y(), false);
280
281     waitForMotorStop();
282
283     this->on_centerCam1pushButton_clicked();
284
285     xL = ui->xActualPoslabel->text().toDouble();
286     yL = ui->yActualPoslabel->text().toDouble();
287
288     XAxis.moveToPosition(sourceTray.getReferencePoint2X(), false);
289     YAxis.moveToPosition(sourceTray.getReferencePoint2Y(), false);
290
291     waitForMotorStop();
292
293     this->on_centerCam1pushButton_clicked();
294
295     xR = ui->xActualPoslabel->text().toDouble();
296     yR = ui->yActualPoslabel->text().toDouble();
297
298     deltaX = xR - xL;
299     deltaY = yR - yL;
300
301     sourceTray.setXCorrectionFactor(deltaY/deltaX);
302 }
303
304
```

```
305 void TestHandler::on_calPasspushButton_clicked()
306 {
307     double deltaX, deltaY, xR, yR, xL, yL;
308
309     XAxis.moveToPosition(passTray.getReferencePoint1X(), false);
310     YAxis.moveToPosition(passTray.getReferencePoint1Y(), false);
311
312     waitForMotorStop();
313
314     this->on_centerCam1pushButton_clicked();
315
316     xL = ui->xActualPoslabel->text().toDouble();
317     yL = ui->yActualPoslabel->text().toDouble();
318
319     XAxis.moveToPosition(passTray.getReferencePoint2X(), false);
320     YAxis.moveToPosition(passTray.getReferencePoint2Y(), false);
321
322     waitForMotorStop();
323
324     this->on_centerCam1pushButton_clicked();
325
326     xR = ui->xActualPoslabel->text().toDouble();
327     yR = ui->yActualPoslabel->text().toDouble();
328
329     deltaX = xR - xL;
330     deltaY = yR - yL;
331
332     passTray.setXCorrectionFactor(deltaY/deltaX);
333 }
334
335
336
337 void TestHandler::on_cropImagepushButton_clicked()
338 {
339     Mat freshImage1;
340     source2.operator >> (freshImage1);
341     imshow("freshImage", freshImage1);
342
343     Mat image2Temp;
344     freshImage1.copyTo(image2Temp);
345     flip(image2Temp, freshImage1, 1);
346
347     rotateImg(freshImage1, cam2CorDeg, Point(freshImage1.cols/2.0, freshImage1.
          rows/2.0));
348     imshow("rot␣dreshImage1", freshImage1);
349
350     Mat crop= cropChip(freshImage1);
351     imshow("cropImg", crop);
352     compareImage(crop, chipTemplate);
353 }
354
355 void TestHandler::on_gotoCamerapushButton_clicked()
356 {
357     goToCamera();
358 }
359
360 void TestHandler::on_gotoTestSocketpushButton_clicked()
361 {
362     goToTestSocket();
363 }
364
```

```
365 void TestHandler::on_failTraypushButton_clicked()
366 {
367     goToFailTray();
368 }
369
370 void TestHandler::on_sourceTraypushButton_clicked()
371 {
372     goToSourceTray();
373 }
374
375 void TestHandler::on_passTraypushButton_clicked()
376 {
377     goToPassTray();
378 }
379
380 void TestHandler::on_centerChippushButton_clicked()
381 {
382     XAxis.moveToPosition(this->cameraUpXpos, false);
383     YAxis.moveToPosition(this->cameraUpYpos, false);
384     waitForMotorStop();
385
386     adjustChip();
387 }
388
389
390 void TestHandler::on_SourcegoToChippushButton_clicked()
391 {
392     sourceTray.returnToZero();
393     sourceTray.goToChipByCoordinate(ui->sourceRow->text().toDouble(), ui->
          sourceCol->text().toDouble());
394 }
395
396 void TestHandler::on_SourcetoZeropushButton_clicked()
397 {
398     sourceTray.returnToZero();
399 }
400
401 void TestHandler::on_PassgoToChippushButton_clicked()
402 {
403     passTray.returnToZero();
404     passTray.goToChipByCoordinate(ui->passRow->text().toDouble(), ui->passCol->
          text().toDouble());
405 }
406
407 void TestHandler::on_PasstoZeropushButton_clicked()
408 {
409     passTray.returnToZero();
410 }
411
412 void TestHandler::on_FailgoToChippushButton_clicked()
413 {
414     failTray.returnToZero();
415     failTray.goToChipByCoordinate(ui->failRow->text().toDouble(), ui->failCol->
          text().toDouble());
416 }
417
418 void TestHandler::on_FailtoZeropushButton_clicked()
419 {
420     failTray.returnToZero();
421 }
422
```

```
423 void TestHandler::on_calalltrayspushButton_clicked()
424 {
425     calAllTrays();
426 }
427
428 void TestHandler::on_startAutomatedTestpushButton_clicked()
429 {
430     //Für Messreihen FSM
431     this->state = 0;
432
433     this->measureYCompansation();
434     waitForMotorStop();
435
436     calAllTrays();
437     waitForMotorStop();
438
439     stopFSM = false;
440     this->testFSM();
441 }
442
443 void TestHandler::on_stopAutomatedTestpushButton_clicked()
444 {
445     stopFSM = true;
446     state = 0;
447 }
448
449 void TestHandler::on_resetCounterpushButton_clicked()
450 {
451     ui->passLabel->setNum(0);
452     ui->failLabel->setNum(0);
453
454     ui->progressBar->setValue(0);
455     ui->resultLabel->setText("Result");
456     ui->resultLabel->setStyleSheet("");
457 }
458
459
460 /*****************************************************
461  *
462  *
463  *  Tab 2 buttons and other control input
464  *
465  *
466 *****************************************************/
467
468 void TestHandler::on_updateModulesButton_clicked()
469 {
470 #ifdef DEBUG_ON
471 #ifdef DEBUG_ON
472     qDebug() << __METHOD_NAME__;
473 #endif
474 #endif
475     updateModuleList();
476 }
477
478 //saves the choosen modules for the axis and initializes them
479 void TestHandler::on_saveModuleListButton_clicked()
480 {
481     saveModuleList();
482
483     if((ui->XModulecomboBox->currentText() != "<none>") &&
484             (ui->YModulecomboBox->currentText() != "<none>") &&
485             (ui->ZModulecomboBox->currentText() != "<none>") &&
486             (ui->EModulecomboBox->currentText() != "<none>"))
487     {
488         //initialize each axis with a default parameters so it can move
489         XAxis.init();
490         YAxis.init();
491         ZAxis.init();
492         EAxis.init();
493
494         updateModuleSettingsGUI();
495     }
496 }
497
498 void TestHandler::on_loadSettingspushButton_clicked()
499 {
500     loadHardwareSettings();
501 }
502
503 void TestHandler::on_loadChipSettingspushButton_clicked()
504 {
505     loadChipSettings();
506 }
507
508 void TestHandler::on_loadChipTemplatepushButton_clicked()
509 {
510     loadChipTemplate();
511 }
512
513 /*****************************************************
514  *
515  *
516  *  GUI specific functions
517  *
518  *
519 *****************************************************/
520 void TestHandler::initComboBoxes()
521 {
522     ui->XMotorDirectioncomboBox->insertItems(0,QStringList() << "Yes" << "No");
523     ui->YMotorDirectioncomboBox->insertItems(0,QStringList() << "Yes" << "No");
524     ui->ZMotorDirectioncomboBox->insertItems(0,QStringList() << "Yes" << "No");
525     ui->EMotorDirectioncomboBox->insertItems(0,QStringList() << "Yes" << "No");
526
527     ui->XSwitchEndstopscomboBox->insertItems(0,QStringList() << "No" << "Yes");
528     ui->YSwitchEndstopscomboBox->insertItems(0,QStringList() << "No" << "Yes");
529     ui->ZSwitchEndstopscomboBox->insertItems(0,QStringList() << "No" << "Yes");
530
531     ui->XhomingDirectioncomboBox->insertItems(0,QStringList() << "Left" << "
            Right");
532     ui->YhomingDirectioncomboBox->insertItems(0,QStringList() << "Left" << "
            Right");
533     ui->ZhomingDirectioncomboBox->insertItems(0,QStringList() << "Left" << "
            Right");
534 }
535
536 void TestHandler::updateModuleSettingsGUI()
537 {
538     //Set the GUI Controls corresponding to the .ini file
539     ui->XAxiscomboBox->setCurrentIndex(XAxis.getAxis());
540     ui->YAxiscomboBox->setCurrentIndex(YAxis.getAxis());
541     ui->ZAxiscomboBox->setCurrentIndex(ZAxis.getAxis());
```

```
542     ui->EAxiscomboBox->setCurrentIndex(EAxis.getAxis());
543
544     if(XAxis.getHomingDirection())
545         ui->XhomingDirectioncomboBox->setCurrentText("Left");
546     else
547         ui->XhomingDirectioncomboBox->setCurrentText("Right");
548
549     if(YAxis.getHomingDirection())
550         ui->YhomingDirectioncomboBox->setCurrentText("Left");
551     else
552         ui->YhomingDirectioncomboBox->setCurrentText("Right");
553
554     if(ZAxis.getHomingDirection())
555         ui->ZhomingDirectioncomboBox->setCurrentText("Left");
556     else
557         ui->ZhomingDirectioncomboBox->setCurrentText("Right");
558
559
560     if(XAxis.getSwitchEndstops())
561         ui->XSwitchEndstopscomboBox->setCurrentText("Yes");
562     else
563         ui->XSwitchEndstopscomboBox->setCurrentText("No");
564
565     if(YAxis.getSwitchEndstops())
566         ui->YSwitchEndstopscomboBox->setCurrentText("Yes");
567     else
568         ui->YSwitchEndstopscomboBox->setCurrentText("No");
569
570     if(ZAxis.getSwitchEndstops())
571         ui->ZSwitchEndstopscomboBox->setCurrentText("Yes");
572     else
573         ui->ZSwitchEndstopscomboBox->setCurrentText("No");
574
575
576     if(XAxis.getinverseMotor())
577         ui->XMotorDirectioncomboBox->setCurrentText("Yes");
578     else
579         ui->XMotorDirectioncomboBox->setCurrentText("No");
580
581     if(YAxis.getinverseMotor())
582         ui->YMotorDirectioncomboBox->setCurrentText("Yes");
583     else
584         ui->YMotorDirectioncomboBox->setCurrentText("No");
585
586     if(ZAxis.getinverseMotor())
587         ui->ZMotorDirectioncomboBox->setCurrentText("Yes");
588     else
589         ui->ZMotorDirectioncomboBox->setCurrentText("No");
590
591     if(EAxis.getinverseMotor())
592         ui->EMotorDirectioncomboBox->setCurrentText("Yes");
593     else
594         ui->EMotorDirectioncomboBox->setCurrentText("No");
595 }
596
597 void TestHandler::changeResultLabel(int result)
598 {
599     ui->resultLabel->setText("Finished:␣" + QString::number(result));
600
601
602     //the random generator puts out numbers between 1 and 10
603     //Two numbers, 1 and 6, represent a faulty chip.
604     //So statistically two chips fail and eight chips pass.
605     //That makes a ratio of 1/4
606     if((result != 1) && (result != 6))
607     {
608         ui->resultLabel->setStyleSheet("QLabel␣{␣background-color␣:␣green;␣␣}");
609         ui->passLabel->setNum( ui->passLabel->text().toInt() + 1);
610         testResult = true;
611     }
612     else
613     {
614         ui->resultLabel->setStyleSheet("QLabel␣{␣background-color␣:␣red;␣␣}");
615         ui->failLabel->setNum( ui->failLabel->text().toInt() + 1);
616         testResult = false;
617     }
618 }
619
620 void TestHandler::errorMessage(QString errorCode)
621 {
622     qDebug()<<"error␣message";
623     QMessageBox::information(this,"ERROR", "Error␣Code:\t"+errorCode);
624     ui->resultLabel->setText("Result");
625     ui->resultLabel->setStyleSheet("");
626 }
```

Listing E.9: Quellcode von StateMachines.cpp

```
1  #include "TestHandler.h"
2  #include <QtGui>
3  #include "ui_TestHandler.h"
4  #include <chrono>
5  #include <thread>
6
7  void TestHandler::testFSM()
8  {
9      while(stopFSM == false)
10     {
11         QApplication::processEvents();
12
13         switch(state)
14         {
15
16         case 0:
17             sourceTray.returnToZero();
18             state = 1;
19             break;
20
21         case 1:
22             if(sourceTray.goToNextChip())
23                 state = 2;
24             else
25                 stopFSM = true;
26             break;
27
28         case 2:
29             sourceTray.getChip();
30             state = 3;
31             break;
32
```

```
33          case 3:
34              goToCamera();
35              state = 4;
36              break;
37
38          case 4:
39              this->correctPosition = adjustChip();
40              if(correctPosition)
41                  state = 5;
42              else
43              {
44                  if(vacPipetteEmpty)
45                      state = 17;
46                  else
47                      state = 14;
48
49                  ui->failLabel->setNum( ui->failLabel->text().toInt() + 1);
50              }
51              break;
52
53          case 5:
54              goToTestSocket();
55              state = 6;
56              break;
57
58          case 6:
59              XAxis.moveToPosition( -(this->dX), true);
60              YAxis.moveToPosition(-(this->dY), true);
61              waitForMotorStop();
62              state = 7;
63              break;
64
65          case 7:
66              XAxis.moveToPosition(this->CameraDownDeltaX, true);
67              YAxis.moveToPosition(this->CameraDownDeltaY, true);
68              waitForMotorStop();
69              state = 8;
70              break;
71
72          case 8:
73              ZAxis.velocityMode(-7);
74              waitForMotorStop();
75              state = 9;
76              break;
77
78          case 9:
79              //Testing...
80              startTestSocket();
81              while(!testSocket.isFinished())
82              {
83                  QApplication::processEvents();
84              }
85              state = 10;
86              break;
87
88          case 10:
89              ZAxis.velocityMode(10);
90              waitForMotorStop();
91              if(testResult)
92                  state = 11;
93              else
94                  state = 14;
95              break;
96
97          case 11:
98              passTray.returnToZero();
99              state = 12;
100             break;
101
102         case 12:
103             if(passTray.goToNextChip())
104             {
105                 double temp;
106                 switch(passTray.getRotationForPlacing())
107                 {
108                 case 0:     //0 degrees
109                     break;
110
111                 case 90:    //90 degrees
112                     temp = this->dX;
113                     this->dX = -(this->dY);
114                     this->dY = temp;
115                     break;
116
117                 case 180:   //180 degrees
118                     temp = this->dX;
119                     this->dX = -(this->dX);
120                     this->dY = -(temp);
121                     break;
122
123                 case -90:   //-90 degrees
124                     temp = this->dX;
125                     this->dX = this->dY;
126                     this->dY = -(temp);
127                     break;
128
129                 default:
130                     break;
131                 }
132
133                 std::this_thread::sleep_for(std::chrono::milliseconds(100));
134                 XAxis.moveToPosition( -(this->dX), true);
135                 YAxis.moveToPosition(-(this->dY), true);
136                 waitForMotorStop();
137                 std::this_thread::sleep_for(std::chrono::milliseconds(100));
138             }
139             else
140                 stopFSM = true;
141             state = 13;
142             break;
143
144
145         case 13:
146             passTray.placeChip();
147             state = 0;
148             break;
149
150
151             //From here for failed chips
152         case 14:
153             failTray.returnToZero();
154
```

```
155             state = 15;
156             break;
157
158         case 15:
159             if(failTray.goToNextChip())
160             {
161                 double temp;
162                 switch(failTray.getRotationForPlacing())
163                 {
164                 case 0:     //0 degrees
165                     break;
166
167                 case 90:    //90 degrees
168                     temp = this->dX;
169                     this->dX = -(this->dY);
170                     this->dY = temp;
171                     break;
172
173                 case 180:   //180 degrees
174                     temp = this->dX;
175                     this->dX = -(this->dX);
176                     this->dY = -(temp);
177                     break;
178
179                 case -90:   //-90 degrees
180                     temp = this->dX;
181                     this->dX = this->dY;
182                     this->dY = -(temp);
183                     break;
184
185                 default:
186                     break;
187                 }
188
189                 std::this_thread::sleep_for(std::chrono::milliseconds(100));
190                 XAxis.moveToPosition( -(this->dX), true);
191                 YAxis.moveToPosition(-(this->dY), true);
192                 waitForMotorStop();
193                 std::this_thread::sleep_for(std::chrono::milliseconds(100));
194             }
195             else
196                 stopFSM = true;
197             state = 16;
198             break;
199
200         case 16:
201             failTray.placeChip();
202             state = 0;
203             break;
204
205         case 17:
206             XAxis.moveToPosition( 10, false);
207             YAxis.moveToPosition(10, false);
208             waitForMotorStop();
209             XAxis.setIO(0,1);
210             state = 0;
211             break;
212
213         default:
214             break;
215         }
```

```
216     }
217 }
218
219
220 void TestHandler::calAllTrays()
221 {
222     sourceTray.setXAxis(&XAxis);
223     sourceTray.setYAxis(&YAxis);
224     sourceTray.setZAxis(&ZAxis);
225
226     passTray.setXAxis(&XAxis);
227     passTray.setYAxis(&YAxis);
228     passTray.setZAxis(&ZAxis);
229
230     failTray.setXAxis(&XAxis);
231     failTray.setYAxis(&YAxis);
232     failTray.setZAxis(&ZAxis);
233
234     goToFailTray();
235     waitForMotorStop();
236     failTray.setXZero( ui->xActualPoslabel->text().toDouble());
237     failTray.setYZero( ui->yActualPoslabel->text().toDouble());
238
239     goToSourceTray();
240     waitForMotorStop();
241     sourceTray.setXZero( ui->xActualPoslabel->text().toDouble());
242     sourceTray.setYZero( ui->yActualPoslabel->text().toDouble());
243
244     goToPassTray();
245     waitForMotorStop();
246     passTray.setXZero( ui->xActualPoslabel->text().toDouble());
247     passTray.setYZero( ui->yActualPoslabel->text().toDouble());
248 }
```

Listing E.10: Quellcode von Calibration.cpp

```
1  #include "TestHandler.h"
2  #include <QtGui>
3  #include "ui_TestHandler.h"
4
5
6  //measures and calculates the compensationfactor for
7  //the Y Axis
8  void TestHandler::measureYCompansation()
9  {
10     double deltaX, deltaY, xTL, yTL, xBL, yBL;
11
12     XAxis.moveToPosition(this->RefCoordinateBottomLeftX, false);
13     YAxis.moveToPosition(this->RefCoordinateBottomLeftY, false);
14
15     waitForMotorStop();
16
17     this->on_centerCam1pushButton_clicked();
18
19     xBL = ui->xActualPoslabel->text().toDouble();
20     yBL = ui->yActualPoslabel->text().toDouble();
21
22     XAxis.moveToPosition(this->RefCoordinateTopLeftX, false);
23     YAxis.moveToPosition(this->RefCoordinateTopLeftY, false);
```

```
24
25      waitForMotorStop();
26
27      this->on_centerCam1pushButton_clicked();
28
29      xTL = ui->xActualPoslabel->text().toDouble();
30      yTL = ui->yActualPoslabel->text().toDouble();
31
32      deltaX = xTL - xBL;
33      deltaY = yTL - yBL;
34      correctionFactorY = deltaX/deltaY;
35      failTray.setYCorrectionFactor(correctionFactorY);
36      passTray.setYCorrectionFactor(correctionFactorY);
37      sourceTray.setYCorrectionFactor(correctionFactorY);
38  }
39
40  //Centers the downlooking driving camera above
41  //the middle of a reference marker
42  bool TestHandler::goToCenterCam1()
43  {
44      Mat freshImage1;
45      source1.operator >> (freshImage1);
46
47      Mat image2Temp;
48      freshImage1.copyTo(image2Temp);
49      flip(image2Temp, freshImage1, 1);
50      rotateImg(freshImage1, cam1CorDeg, Point(freshImage1.cols/2.0, freshImage1.
            rows/2.0));
51
52      Mat newBackround;
53      generateEvenBackround(freshImage1, 160,120,320,240).copyTo(newBackround);
54
55      double xCorrection = getDeltaX(newBackround);
56      double yCorrection = getDeltaY(newBackround);
57
58      if((fabs(xCorrection/pixelTommFactorCam1)< 0.05) && (fabs(yCorrection/
            pixelTommFactorCam1)<0.05))
59          return true;
60      else
61      {
62          if((xCorrection >= 0.0) && (yCorrection >= 0.0))
63          {
64              XAxis.moveToPosition((fabs(xCorrection)/ pixelTommFactorCam1), true)
                    ;
65              YAxis.moveToPosition((fabs(yCorrection)/ pixelTommFactorCam1), true)
                    ;
66          }
67          else if((xCorrection <= 0.0) && (yCorrection >= 0.0))
68          {
69              XAxis.moveToPosition((-fabs(xCorrection)/ pixelTommFactorCam1), true
                    );
70              YAxis.moveToPosition((fabs(yCorrection)/ pixelTommFactorCam1), true)
                    ;
71          }
72          else if((xCorrection >= 0.0) && (yCorrection <= 0.0))
73          {
74              XAxis.moveToPosition((fabs(xCorrection)/ pixelTommFactorCam1), true)
                    ;
75              YAxis.moveToPosition((-fabs(yCorrection)/ pixelTommFactorCam1), true
                     );
76          }
77          else if((xCorrection <= 0.0) && (yCorrection <= 0.0))
78          {
79              XAxis.moveToPosition((-fabs(xCorrection)/ pixelTommFactorCam1), true
                     );
80              YAxis.moveToPosition((-fabs(yCorrection)/ pixelTommFactorCam1), true
                    );
81          }
82          return false;
83      }
84  }
85
86  //Centers a chip over the uplooking
87  //microsope camera
88  bool TestHandler::goToCenterCam2()
89  {
90      Mat freshImage1;
91      source2.operator >> (freshImage1);
92
93      Mat image2Temp;
94      freshImage1.copyTo(image2Temp);
95      flip(image2Temp, freshImage1, 1);
96      rotateImg(freshImage1, cam2CorDeg, Point(freshImage1.cols/2.0, freshImage1.
            rows/2.0));
97
98      Mat newBackround;
99      generateEvenBackround(freshImage1, 160,120,320,240).copyTo(newBackround);
100
101     double xCorrection = getDeltaX(newBackround);
102     double yCorrection = getDeltaY(newBackround);
103
104     if((fabs(xCorrection/pixelTommFactorCam2)< 0.05) && (fabs(yCorrection/
            pixelTommFactorCam2)<0.05))
105         return true;
106     else
107     {
108         if((xCorrection >= 0.0) && (yCorrection >= 0.0))
109         {
110             XAxis.moveToPosition((-fabs(xCorrection)/ pixelTommFactorCam2), true
                    );
111             YAxis.moveToPosition((fabs(yCorrection)/ pixelTommFactorCam2), true)
                    ;
112         }
113         else if((xCorrection <= 0.0) && (yCorrection >= 0.0))
114         {
115             XAxis.moveToPosition((fabs(xCorrection)/ pixelTommFactorCam2), true)
                    ;
116             YAxis.moveToPosition((fabs(yCorrection)/ pixelTommFactorCam2), true)
                    ;
117         }
118         else if((xCorrection >= 0.0) && (yCorrection <= 0.0))
119         {
120             XAxis.moveToPosition((-fabs(xCorrection)/ pixelTommFactorCam2), true
                    );
121             YAxis.moveToPosition((-fabs(yCorrection)/ pixelTommFactorCam2), true
                     );
122         }
123         else if((xCorrection <= 0.0) && (yCorrection <= 0.0))
124         {
125             XAxis.moveToPosition((fabs(xCorrection)/ pixelTommFactorCam2), true
                    );
```

```
126             YAxis.moveToPosition((-fabs(yCorrection)/ pixelTommFactorCam2), true
                    );
127         }
128         return false;
129     }
130 }
131
132 //Rotates the picked up chip
133 //so that it is square to the Testhandler
134 void TestHandler::adjustRotation()
135 {
136     Mat freshImage1;
137     source2.operator >> (freshImage1);
138
139     Mat newBackround;
140     generateEvenBackround(freshImage1, 160,120,320,240).copyTo(newBackround);
141
142     double winkel = getDegree(newBackround);
143
144     if(fabs(winkel) < (winkel+90) )
145     {
146         EAxis.moveToPosition(-winkel - cam2CorDeg, true);
147     }
148     else if(fabs(winkel) > (winkel + 90))
149     {
150         EAxis.moveToPosition((-(winkel + 90)) - cam2CorDeg, true);
151     }
152 }
153
154 //This function checks if the chip
155 //above the microscope camera matches the master template
156 //it returns a true if it matched
157 //it returns a flase if not
158 //if it detects that there is no chip at all it
159 //sets the vacPipetteEmpty to true
160 bool TestHandler::adjustChip()
161 {
162     adjustRotation();
163     waitForMotorStop();
164
165     float similarity = 1.0;
166     int  i = 0;
167     while((similarity >= 0.007) && (i <= 8))
168     {
169         //rotate another 90°
170         EAxis.moveToPosition(90, true);
171         waitForMotorStop();
172
173         //center the chip in the middle of the camera
174         bool check = false;
175         while(!check)
176         {
177             check = goToCenterCam2();
178             waitForMotorStop();
179         }
180
181         //crop and compare image
182         Mat freshImage1;
183         source2.operator >> (freshImage1);
184
185         Mat image2Temp;
186         freshImage1.copyTo(image2Temp);
187         flip(image2Temp, freshImage1, 1);
188
189         rotateImg(freshImage1, cam2CorDeg, Point(freshImage1.cols/2.0,
                freshImage1.rows/2.0));
190
191         Mat crop= cropChip(freshImage1);
192         similarity = compareImage(crop, chipTemplate);
193
194         i++;
195     }
196
197     //check if images are similar enough
198     if(similarity <= 0.007)
199     {
200         this->dX = this->cameraUpXpos - (    double(XAxis.actualPosition()) /
                double(XAxis.getStepsPermm())    );
201         this->dY = this->cameraUpYpos - (    double(YAxis.actualPosition()) /
                double(YAxis.getStepsPermm())    );
202         return true;
203     }
204
205     //check if there is no chip at all
206     if(similarity > 0.9)
207     {
208         this->vacPipetteEmpty = true;
209     }
210     else
211         this->vacPipetteEmpty = false;
212
213     this->dX = this->cameraUpXpos - (    double(XAxis.actualPosition()) / double
            (XAxis.getStepsPermm())    );
214     this->dY = this->cameraUpYpos - (    double(YAxis.actualPosition()) / double
            (YAxis.getStepsPermm())    );
215
216     return false;
217 }
```

Listing E.11: Quellcode von Camera.cpp

```
1  #include "TestHandler.h"
2  #include <QtGui>
3  #include "ui_TestHandler.h"
4
5  void TestHandler::startCamera()
6  {
7      qDebug() << __METHOD_NAME__;
8      source1  = VideoCapture(0);
9      if(!source1.isOpened())  // check if we succeeded
10         qDebug() << __METHOD_NAME__ << "Can␣not␣open␣camera␣(0)" ;
11
12     source2  = VideoCapture(1);
13     if(!source2.isOpened())  // check if we succeeded
14         qDebug() << __METHOD_NAME__ << "Can␣not␣open␣camera␣(1)" ;
15
16     scene1 = new QGraphicsScene();
17     scene2 = new QGraphicsScene();
18     timerIDcamera = startTimer(30);//every 30 ms the function timerEvent is
           called
```

```
}

//this method converts a opencv Mat to an QImage
//the code is from:
//http://qtandopencv.blogspot.de/2013/08/how-to-convert-between-cvmat-and-qimage.html
//(15.07.2016)
QImage TestHandler::mat_to_qimage_ref(Mat &mat, QImage::Format format)
{
    return QImage(mat.data, mat.cols, mat.rows, format);
}

void TestHandler::goToCamera()
{
    XAxis.moveToPosition(this->cameraUpXpos, false);
    YAxis.moveToPosition(this->cameraUpYpos, false);
    waitForMotorStop();
}
```

Listing E.12: Quellcode von Tray.h

```
#ifndef TRAY_H
#define TRAY_H

#include <QObject>
#include "AxisConfiguration.h"

class Tray: public QObject   // von QPushButton ableiten -> indirekt von QObject abgeleitet
{

    Q_OBJECT

public:
    Tray();

    //move to the next chip
    //if the tray is full or empty it returns a false
    bool goToNextChip();

    //move to a chip specified by row and column
    void goToChipByCoordinate(int row, int col);

    //move back to the zero point of the tray
    void returnToZero();

    //pick up chip which is currently under the camera
    void getChip();

    //place up chip which is currently under the camera
    void placeChip();

    //wait untill all motors stopped
    //the whiles loops are in an extra thread,
    //so the program is not blocked
    void waitForMotorStop();

    void setXFromCorner(double value);

    void setYFromCorner(double value);

    void setXStep(double value);

    void setYStep(double value);

    void setRows(int value);

    void setColumns(int value);

    void setXCorrectionFactor(double value);

    void setYCorrectionFactor(double value);

    void setVertical(bool value);

    void setXAxis(AxisConfiguration *value);

    void setYAxis(AxisConfiguration *value);

    void setEAxis(AxisConfiguration *value);

    void setZAxis(AxisConfiguration *value);

    void setXZero(double value);

    void setYZero(double value);

    QString getName() const;
    void setName(const QString &value);

    int getRows() const;

    int getColumns() const;

    double getReferencePoint1X() const;
    void setReferencePoint1X(double value);

    double getReferencePoint1Y() const;
    void setReferencePoint1Y(double value);

    double getReferencePoint2X() const;
    void setReferencePoint2X(double value);

    double getReferencePoint2Y() const;
    void setReferencePoint2Y(double value);

    void setCameraDownDeltaX(double value);

    void setCameraDownDeltaY(double value);

    void setMechanicalXCorrectionfactor(double value);

    void setMechanicalYCorrectionfactor(double value);

    int getRotationForPlacing() const;
    void setRotationForPlacing(int value);

    int getCurrentRow() const;

    int getCurrentColumn() const;
```

```
99
100     void setCurrentRow(int value);
101
102     void setCurrentColumn(int value);
103
104 private:
105     QString name;
106     double xFromCorner;
107     double yFromCorner;
108     double xStep;
109     double yStep;
110     int rows;
111     int columns;
112     int currentRow;
113     int currentColumn;
114     double xCorrectionFactor;
115     double yCorrectionFactor;
116     double XZero;
117     double YZero;
118     double referencePoint1X;//Closest to Tray corner
119     double referencePoint1Y;//Closest to Tray corner
120     double referencePoint2X;
121     double referencePoint2Y;
122     double CameraDownDeltaX;
123     double CameraDownDeltaY;
124     bool vertical;
125     AxisConfiguration *XAxis;
126     AxisConfiguration *YAxis;
127     AxisConfiguration *ZAxis;
128     AxisConfiguration *EAxis;
129     double MechanicalXCorrectionfactor;
130     double MechanicalYCorrectionfactor;
131     int RotationForPlacing;
132 };
133
134 #endif // TRAY_H
```

Listing E.13: Quellcode von Tray.cpp

```
1  #include "Tray.h"
2  #include <chrono>
3  #include <thread>
4  #include "ImageProcessing.h"
5
6  Tray::Tray()
7  {   //example init
8      //LQFP 10x10x1.4 eg. TMC 246
9      xFromCorner = 13.0;//cols
10     yFromCorner = 13.1;//rows
11     xStep = 15.7;
12     yStep = 15.2;
13     rows = 8;
14     columns = 19;
15     xCorrectionFactor = 0.0;
16     yCorrectionFactor = 0.0;
17     vertical = true;
18     XZero = 0.0;
19     YZero = 0.0;
20     MechanicalXCorrectionfactor = 0.0;
21     MechanicalYCorrectionfactor = 0.0;
22     RotationForPlacing = 0;
23     currentRow = 1;
24     currentColumn = 1;
25     name = "";
26 }
27
28 //move to the next chip
29 //if the tray is full or empty it returns a false
30 bool Tray::goToNextChip()
31 {
32     double distanceX, distanceY, compensationY, compensationX;
33     double mechCompDistancex, mechCompDistancey;
34     double compensationDistance;
35
36     if(currentColumn > columns)
37     {
38         return false;//Tray is full
39     }
40
41     if(vertical)
42     {
43         distanceX = xFromCorner + ((currentRow - 1) * xStep);
44         distanceY = yFromCorner + ((currentColumn - 1) * yStep);
45         compensationY = -distanceY * yCorrectionFactor;
46         compensationX = distanceX * xCorrectionFactor;
47         mechCompDistancex = distanceX * MechanicalXCorrectionfactor;
48         mechCompDistancey = distanceY * MechanicalYCorrectionfactor;
49         compensationDistance = -(distanceY + mechCompDistancey) *
               yCorrectionFactor;
50         this->XAxis->moveToPosition((distanceX + mechCompDistancex +
               compensationDistance), true);
51         this->YAxis->moveToPosition(-(distanceY + mechCompDistancey) , true);
52     }
53     else
54     {
55         distanceX = xFromCorner + ((currentRow - 1) * xStep);
56         distanceY = yFromCorner + ((currentColumn - 1) * yStep);
57         compensationY = distanceX * yCorrectionFactor;
58         compensationX = distanceX * xCorrectionFactor;
59         mechCompDistancex = distanceX * MechanicalXCorrectionfactor;
60         mechCompDistancey = distanceY * MechanicalYCorrectionfactor;
61         compensationDistance = (distanceX + mechCompDistancex) *
               yCorrectionFactor;
62         this->XAxis->moveToPosition((distanceY + mechCompDistancey +
               compensationDistance), true);
63         this->YAxis->moveToPosition((distanceX + mechCompDistancex) , true);
64     }
65
66     currentRow++;
67     if(currentRow > rows)
68     {
69         currentRow = 1;
70         currentColumn++;
71     }
72     this->waitForMotorStop();
73     return true;
74 }
75
76 //move to a chip specified by row and column
77 void Tray::goToChipByCoordinate(int row, int col)
```

```
78  {
79      double distanceX, distanceY, compensationY, compensationX;
80      double mechCompDistancex, mechCompDistancey;
81      double compensationDistance;
82
83      if(vertical)
84      {
85          distanceX = xFromCorner + ((row - 1) * xStep);
86          distanceY = yFromCorner + ((col - 1) * yStep);
87          compensationY = -distanceY * yCorrectionFactor;
88          compensationX = distanceX * xCorrectionFactor;
89          mechCompDistancex = distanceX * MechanicalXCorrectionfactor;
90          mechCompDistancey = distanceY * MechanicalYCorrectionfactor;
91          compensationDistance = -(distanceY + mechCompDistancey) *
                yCorrectionFactor;
92          this->XAxis->moveToPosition((distanceX + mechCompDistancex +
                compensationDistance), true);
93          this->YAxis->moveToPosition(-(distanceY + mechCompDistancey) , true);
94      }
95      else
96      {
97          distanceX = xFromCorner + ((row - 1) * xStep);
98          distanceY = yFromCorner + ((col - 1) * yStep);
99          compensationY = distanceX * yCorrectionFactor;
100         compensationX = distanceX * xCorrectionFactor;
101         mechCompDistancex = distanceX * MechanicalXCorrectionfactor;
102         mechCompDistancey = distanceY * MechanicalYCorrectionfactor;
103         compensationDistance = (distanceX + mechCompDistancex) *
                yCorrectionFactor;
104         this->XAxis->moveToPosition((distanceY + mechCompDistancey +
                compensationDistance), true);
105         this->YAxis->moveToPosition((distanceX + mechCompDistancex) , true);
106     }
107     this->waitForMotorStop();
108 }
109
110 //move back to the zero point of the tray
111 void Tray::returnToZero()
112 {
113     this->XAxis->moveToPosition(XZero, false);
114     this->YAxis->moveToPosition(YZero, false);
115     this->waitForMotorStop();
116 }
117
118 //pick up chip which is currently under the camera
119 void Tray::getChip()
120 {
121     XAxis->moveToPosition(CameraDownDeltaX, true);
122     YAxis->moveToPosition(CameraDownDeltaY,true);
123     this->waitForMotorStop();
124
125     ZAxis->velocityMode(-10);
126     std::this_thread::sleep_for(std::chrono::milliseconds(50));
127     while(ZAxis->actualSpeed() != 0)
128     {
129         QApplication::processEvents();
130     }
131
132     this->XAxis->setIO(0,0);//Vacuum on
133
134     std::this_thread::sleep_for(std::chrono::milliseconds(200));
135
136     ZAxis->moveToPosition(-5.0, false);
137     std::this_thread::sleep_for(std::chrono::milliseconds(50));
138     while(ZAxis->actualSpeed() != 0)
139     {
140         QApplication::processEvents();
141     }
142
143     XAxis->moveToPosition(-CameraDownDeltaX, true);
144     YAxis->moveToPosition(-CameraDownDeltaY,true);
145     this->waitForMotorStop();
146 }
147
148 //place up chip which is currently under the camera
149 void Tray::placeChip()
150 {
151     XAxis->moveToPosition(CameraDownDeltaX, true);
152     YAxis->moveToPosition(CameraDownDeltaY,true);
153     EAxis->moveToPosition(double(RotationForPlacing), true);
154     //specifies in which orientation the chip is placed in the tray
155
156     this->waitForMotorStop();
157
158     XAxis->setIO(0,0);
159     ZAxis->velocityMode(-10);
160     std::this_thread::sleep_for(std::chrono::milliseconds(50));
161     while(ZAxis->actualSpeed() != 0)
162     {
163         QApplication::processEvents();
164     }
165     std::this_thread::sleep_for(std::chrono::milliseconds(200));
166     XAxis->setIO(0,1);
167     std::this_thread::sleep_for(std::chrono::milliseconds(1000));
168
169     int vt = ZAxis->getVT();
170     ZAxis->setVT(3000);
171     ZAxis->moveToPosition(1, true);
172     waitForMotorStop();
173     XAxis->moveToPosition(-1, true);
174     waitForMotorStop();
175     ZAxis->setVT(vt);
176
177     ZAxis->moveToPosition(0, false);
178     while(ZAxis->actualSpeed() != 0)
179     {
180         QApplication::processEvents();
181     }
182
183     XAxis->moveToPosition(1, true);
184     //drive the zaxis out of the switching zone
185     ZAxis->moveToPosition(-5.0, false);
186     while(ZAxis->actualSpeed() != 0)
187     {
188         QApplication::processEvents();
189     }
190
191     XAxis->moveToPosition(-CameraDownDeltaX, true);
192     YAxis->moveToPosition(-CameraDownDeltaY,true);
193 }
194
195 //wait untill all motors stopped
```

```
196 //the whiles loops are in an extra thread,
197 //so the program is not blocked
198 void Tray::waitForMotorStop()
199 {
200     while(this->YAxis->actualSpeed() != 0 )
201     {
202         QApplication::processEvents();
203     }
204     while(this->XAxis->actualSpeed() != 0 )
205     {
206         QApplication::processEvents();
207     }
208     while(this->ZAxis->actualSpeed() != 0 )
209     {
210         QApplication::processEvents();
211     }
212     while(this->EAxis->actualSpeed() != 0 )
213     {
214         QApplication::processEvents();
215     }
216 }
217
218 void Tray::setXFromCorner(double value)
219 {
220     xFromCorner = value;
221 }
222
223 void Tray::setYFromCorner(double value)
224 {
225     yFromCorner = value;
226 }
227
228 void Tray::setXStep(double value)
229 {
230     xStep = value;
231 }
232
233 void Tray::setYStep(double value)
234 {
235     yStep = value;
236 }
237
238 void Tray::setRows(int value)
239 {
240     rows = value;
241 }
242
243 void Tray::setColumns(int value)
244 {
245     columns = value;
246 }
247
248 void Tray::setXCorrectionFactor(double value)
249 {
250     xCorrectionFactor = value;
251 }
252
253 void Tray::setYCorrectionFactor(double value)
254 {
255     yCorrectionFactor = value;
256 }
257
258 //Zeropoint is topleft if vertical
259 //and bottomleft if horizontale
260 void Tray::setVertical(bool value)
261 {
262     vertical = value;
263 }
264
265 void Tray::setXAxis(AxisConfiguration *value)
266 {
267     XAxis = value;
268 }
269
270 void Tray::setYAxis(AxisConfiguration *value)
271 {
272     YAxis = value;
273 }
274
275 void Tray::setXZero(double value)
276 {
277     XZero = value;
278 }
279
280 void Tray::setYZero(double value)
281 {
282     YZero = value;
283 }
284
285 void Tray::setZAxis(AxisConfiguration *value)
286 {
287     ZAxis = value;
288 }
289
290 QString Tray::getName() const
291 {
292     return name;
293 }
294
295 void Tray::setName(const QString &value)
296 {
297     name = value;
298 }
299
300 int Tray::getRows() const
301 {
302     return rows;
303 }
304
305 int Tray::getColumns() const
306 {
307     return columns;
308 }
309
310 double Tray::getReferencePoint1X() const
311 {
312     return referencePoint1X;
313 }
314
315 void Tray::setReferencePoint1X(double value)
316 {
317     referencePoint1X = value;
```

```
}

double Tray::getReferencePoint2X() const
{
    return referencePoint2X;
}

void Tray::setReferencePoint2X(double value)
{
    referencePoint2X = value;
}

double Tray::getReferencePoint2Y() const
{
    return referencePoint2Y;
}

void Tray::setReferencePoint2Y(double value)
{
    referencePoint2Y = value;
}

void Tray::setCameraDownDeltaX(double value)
{
    CameraDownDeltaX = value;
}

void Tray::setCameraDownDeltaY(double value)
{
    CameraDownDeltaY = value;
}

void Tray::setMechanicalXCorrectionfactor(double value)
{
    MechanicalXCorrectionfactor = value;
}

void Tray::setMechanicalYCorrectionfactor(double value)
{
    MechanicalYCorrectionfactor = value;
}

int Tray::getRotationForPlacing() const
{
    return RotationForPlacing;
}

void Tray::setRotationForPlacing(int value)
{
    RotationForPlacing = value;
}

void Tray::setEAxis(AxisConfiguration *value)
{
    EAxis = value;
}

int Tray::getCurrentRow() const
{
    return currentRow;
}

int Tray::getCurrentColumn() const
{
    return currentColumn;
}

void Tray::setCurrentRow(int value)
{
    currentRow = value;
}

void Tray::setCurrentColumn(int value)
{
    currentColumn = value;
}

double Tray::getReferencePoint1Y() const
{
    return referencePoint1Y;
}

void Tray::setReferencePoint1Y(double value)
{
    referencePoint1Y = value;
}
```

Listing E.14: Quellcode von TestSocketDummy.h

```
#ifndef TESTSOCKETDUMMY_H
#define TESTSOCKETDUMMY_H

#include <QObject>
#include <QDebug>
#include <QTimerEvent>
#include <QThread>
#include <QTimer>
#include <ctime>

class TestSocketDummy: public QThread
{
    Q_OBJECT

public:
    TestSocketDummy();
    bool isBusy();

    bool setName(const QString &value);

public slots:
    void startTest();

signals:
    void finished(int result);
    void error(QString errorCode);
    void state(int actual);
    void stateMax(int max);

private:
    QString name;
```

```
32      QString TestSocketChipName;
33      bool busy;
34      int result;
35      int numberOfStates;
36      int counter;
37
38  protected :
39      void run();
40  };
41
42  #endif // TESTSOCKETDUMMY_H
```

Listing E.15: Quellcode von TestSocketDummy.cpp

```
1   #include "TestSocketDummy.h"
2   #include <QTimer>
3
4   TestSocketDummy::TestSocketDummy()
5   {
6       //Name of the chip the specific socket ist made for
7       TestSocketChipName = "TMC_5041";
8       numberOfStates = 4;
9       busy = false;
10      counter = 0;
11  }
12
13  //Emulate the Chiptest
14  void TestSocketDummy::startTest()
15  {
16      busy = true;
17      counter = 0;
18
19      emit stateMax(numberOfStates);
20      emit state(counter);
21
22      //Check if the to-be-tested chip has the correct name
23      if(name != TestSocketChipName)
24      {
25          busy = false;
26          emit error("Wrong Chip "+name);
27      }
28      else
29      {
30          this->name = name;
31
32          //pseudo waiting until test complete
33          emit state(0);
34          msleep(500);
35          emit state(1);
36          msleep(500);
37          emit state(2);
38          msleep(500);
39          emit state(3);
40          msleep(500);
41          emit state(4);
42          busy = false;
43
44          srand(time(NULL));
45          result = rand();
46          result = rand();
47          result = rand() % 10;
48          //the random function returns
49          //a value between 1 and 10
50          //
51
52          emit finished(result);
53      }
54  }
55
56  bool TestSocketDummy::setName(const QString &value)
57  {
58      if(value == TestSocketChipName)
59      {
60          name = value;
61          return true;
62      }
63      else
64      {
65          name = value;
66          return false;
67      }
68      return false;
69  }
70
71  //checks if the socket is ready for testing
72  bool TestSocketDummy::isBusy()
73  {
74      return busy;
75  }
76
77  //starts the test
78  void TestSocketDummy::run()
79  {
80      qDebug()<<"run";
81      this->startTest();
82  }
```

F. Inhalt der CD

Auf der beigefügten CD befindet sich:

- Eine PDF-Datei dieser Arbeit
- Der entwickelte C++ Quellcode
- Erwähnte Datenblätter und andere Quellen
- Konfigurationsdateien
- Konstruktionszeichnungen

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 14. September 2016	
Ort, Datum	Unterschrift