HAW HAMBURG

MASTERTHESIS
Chris Niklas Lucka

Prototypische Softwareentwicklung eines Automatisierungssystems unter Einbindung eines neuronalen Netzes zur Objekterkennung

FAKULTÄT TECHNIK UND INFORMATIK
Department Informations- und Elektrotechnik

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

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Chris Niklas Lucka

Prototypische Softwareentwicklung eines Automatisierungssystems unter Einbindung eines neuronalen Netzes zur Objekterkennung

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Automatisierung*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Heike Neumann
Zweitgutachter: Dipl. -Ing. Axel Held (extern)

Eingereicht am: 02.03.2021

Chris Niklas Lucka

Thema der Arbeit

Prototypische Softwareentwicklung eines Automatisierungssystems unter Einbindung eines neuronalen Netzes zur Objekterkennung

Stichworte

Python, PyQt, Neuronale Netze, Objekterkennung, Bildverarbeitung, Tensorflow Object Detection API, Halbleiter, Automatisierung

Kurzzusammenfassung

In dieser Masterthesis wird mit der Tensorflow Object Detection API ein neuronales Netz auf die optische Erkennung von vereinzelten oder sich auf einem Wafer befindenden kleinstelektronischen Bauteilen trainiert und abschließend evaluiert. Hierzu wird zu einem existierenden Hardwarekonzept die softwareseitige Automatisierungslösung entwickelt in der die erwähnte API integriert wird.

Chris Niklas Lucka

Title of Thesis

Prototypical software development of an automation system with integration of a neural network for object detection

Keywords

Python, PyQT, neural network, object detection, image processing, Tensorflow Object Detection API, semiconductor, automation

Abstract

In this masterthesis, a neural network for object detection of electronic devices, which are scattered or located on a wafer is trained and evaluated with the Tensorflow Object Detection API. For an existing hardware concept the software automation solution is developed in which the mentioned API is integrated.

Danksagung

Hiermit bedanke ich mich bei meinen Kolleginnen und Kollegen von Nexperia Gemany GmbH für die zahlreiche Unterstützung während der gesamten Masterarbeit und darüber hinaus. Dazu gehören zum Einen meine technischen Unterstützer, die mir vor allem die Systeme vor Ort erklärt haben, wie Lars Koch, Andreas Zimmermann, Ayk Hilbrink, Mark Schneider und Nils Duchow sowie meine Kontrollleser*innen wie Saskia Riegel, Lars Koch und Nils Duchow. Für diese tatkräftige Unterstützung möchte ich mich bei euch bedanken. Außerdem möchte ich mich bei Axel Held bedanken, der diese Masterarbeit überhaupt erst möglich gemacht hat, da die initiale Idee für die Automatisierung von Axel kam. Als Gruppenleiter hat Axel meine Betreuung nach meiner vollsten Zufriedenheit durchgeführt und konnte mich ebenfalls durch technische Unterstützung fördern. Dafür vielen Dank an dieser Stelle. Ein weiteres Dankeschön geht an Ingolf Jährig, der während der Masterarbeit als mein Mentor agiert hat und diese Aufgabe ebenfalls nach meiner vollsten Zufriedenheit umgesetzt hat.

Des Weiteren möchte ich mich für die intensive Betreuung von Frau Neumann bedanken. Durch die einzelnen Gespräche, die wir hatten, konnte ich die Masterarbeit hervorragend umsetzen und mit Ihnen auf Augenhöhe diskutieren. Sie haben die Hochschule in allen Punkten perfekt vertreten und hatten immer ein offenes Ohr für mich.

Abschließend möchte ich mich bei meiner Familie und meinen Freunden für die Unterstützung während der Masterthesis und dem gesamten Masterstudium bedanken.

Hamburg, 02.03.2021

Inhaltsverzeichnis

Abbildungsverzeichnis **ix**

Tabellenverzeichnis **xii**

Abkürzungen **xiii**

Quelltextverzeichnis **xv**

1 Einleitung **1**

2 Grundlagen **3**

2.1 Halbleiterelektronik . . . 3

2.1.1 DUT . . . 3

2.1.2 Reticle . . . 4

2.1.3 Wafer . . . 6

2.2 Verfügbare Hardware . . . 9

2.2.1 Chuck . . . 11

2.2.2 Semiconductor Parameter Analyzer . . . 12

2.2.3 Beleuchtung . . . 14

2.2.4 Laborrechner . . . 15

2.3 Software . . . 16

2.3.1 Python . . . 16

2.3.2 Python-Paket zur Kommunikation - pyvisa . . . 17

2.3.3 Python-Paket zur Kamera - pypylon . . . 17

2.3.4 Python-Paket zur Benutzeroberfläche - PyQt . . . 18

2.4 Neuronale Netze . . . 18

2.4.1 Training . . . 20

2.4.2 Bildverarbeitung . . . 21

2.4.3 Faster R-CNN . . . 24

3 Aufgabenstellung **26**
3.1 Zielbeschreibung 27

4 Konzept **29**
4.1 SPA 30
4.2 Chuck 30
4.3 Kamera 30
4.4 Licht 31
4.5 Benutzerschnittstelle 31
4.6 Bildverarbeitung 33
4.6.1 Training 33
4.6.2 Wafer 34
4.6.3 Freiliegende DUT 36

5 Implementierung **39**
5.1 Ansteuerung des SPAs 39
5.1.1 VISA-Kommunikation 39
5.1.2 Klasse *KeysightB1500A* 42
5.1.3 Ablauf 1 - Einstellen des SPAs 43
5.1.4 Ablauf 2 - Durchführen der Messungen mit dem SPA 44
5.2 Steuerung des Chucks 46
5.2.1 VISA-Kommunikation 46
5.2.2 Klasse ChuckControl 48
5.2.3 Kontaktierung des Wafers über Stepping 50
5.2.4 Kontaktierung vereinzelter DUT 52
5.2.5 Initialisierung des Chucks 53
5.3 Kamerasteuerung 54
5.4 Lichtsteuerung 55
5.5 Benutzerschnittstelle - GUI 55
5.5.1 Menüleiste 56
5.5.2 Registerauswahl Measurement 60
5.5.3 Registerauswahl Wafermap 63
5.5.4 Funktionsbeschreibung der verknüpften Module in den Applikationen 66
5.5.5 Softwarestruktur der Gesamtanwendung 67

6 Erstellen der Objekterkennung mit der *Tensorflow Object Detection API* 69
6.1 Installation der *Tensorflow Object Detection API* . . . 69
6.2 Training der *Tensorflow Object Detection API* auf die optische Erkennung von DUT . . . 70
6.2.1 Festlegung der Klassifikationen . . . 70
6.2.2 Erstellung des Trainingsdatensatzes . . . 72
6.2.3 Durchführen des Trainings . . . 80
6.3 Evaluation der Trainingsergebnisse . . . 83
6.3.1 Allgemeine Bewertung eines Bildverarbeitungssystems zur Objekterkennung . . . 83
6.3.2 Ausführen der automatischen Evaluierung . . . 85
6.3.3 Vereinzelte DUT . . . 86
6.4 Einbinden des Bildverarbeitungssystems in das Gesamtprogramm . . . 93
6.4.1 Wafer . . . 93
6.4.2 Vereinzelte DUT . . . 95

7 Testen 97
7.1 Testplan . . . 97
7.2 Stabilitätstest . . . 98
7.3 Langzeittest . . . 98

8 Reflexion 99
8.1 Objekterkennung als Bildverarbeitungssystem . . . 99
8.2 Gesamtanwendung . . . 100

9 Ausblick 102

Literaturverzeichnis 104

A Dokumentation - Lastenheft 108

B Dokumentation - Pflichtenheft 109

C Dokumentation - Aufgabenplan 110

D Projektpfad Erläuterung 111

E Einstellung der Hardware für eine lauffähige Objekterkennung 114
E.1 Wafer 114
E.2 Vereinzelte DUT 118

F Informationen zur beigefügten DVD 119

Selbstständigkeitserklärung 120

Abbildungsverzeichnis

2.1 Beispiele für DUT 4
2.2 Aufbau eines Reticles 5
2.3 Exemplarischer 8-Zoll Wafer mit Notch 6
2.4 Exemplarisches Bild vereinzelter DUT 7
2.5 Koordinatensystem auf dem Wafer 8
2.6 Hardwareaufbau des Waferprobers in der Darkbox 10
2.7 Koordinatensystem des Chucks 11
2.8 Semiconductor Parameter Analyzer von Keysight, übernommen aus [34] . 12
2.9 Messspitzen am Darkboxprober 13
2.10 Exemplarischer Wafer unter der Koaxial-Beleuchtung 14
2.11 Exemplarischer Wafer unter LED Licht 15
2.12 Hardwareaufbau mit Laborrechner 15
2.13 Das Perzeptron, modifiziert aus [29] 18
2.14 Die Sigmoid-Funktion (Python Plot) 19
2.15 Exemplarisches neuronales Netz, modifiziert aus [29] 19
2.16 Der Aufbau eines Trainings von neuronalen Netzen 20
2.17 Eingabe eines Bildes in ein neuronales Netz mit exemplarischer Pixelstruktur, modifiziert aus [35] 22
2.18 Objekterkennung in einem Bild, modifiziert aus [35] 23
2.19 Schematische Funktionsweise des Faster R-CNN, übernommen aus [31] . . 24

3.1 Existierender Messablauf für Diodenkennlinien 26
3.2 Ziel der Automatisierung des Darkboxprobers 27

4.1 Konzept der Automatisierung des Darkboxprobers 29
4.2 Konzept der GUI Teil 1 31
4.3 Konzept der GUI Teil 2 32
4.4 Konzept des Trainings des neuronalen Netzes 33
4.5 Fehlkontaktierung der Messspitzen auf dem Wafer 34

4.6 Kontaktierung nach der Korrektur auf dem Wafer 36
4.7 Kontaktierung der freiliegenden DUT 37
4.8 Aufgereihte DUT für die automatisierte Messung 38

5.1 Klassendiagramm zum Modul *spa_hardware.py* 42
5.2 Ablaufdiagramm der Startkommunikation mit dem SPA 43
5.3 Durchführung der einzelnen Messungen am DUT 45
5.4 Klassendiagramm der Klasse *ChuckControl* zum Modul *chuck_hardware.py* 48
5.5 Ablaufdiagramm der Methode *reference_move_position_xy* 49
5.6 Klassendiagramm der Klasse *ResponseThreadChuck* zum Modul *chuck_-hardware.py* 50
5.7 Ablaufdiagramm zum Stepping mit dem Chuck inklusive der Korrektur über Objekterkennung 51
5.8 Ablaufdiagramm der automatisierten Kontaktierung von vereinzelten DUT 52
5.9 Ablaufdiagramm der Initialisierung des Chucks 53
5.10 Klassendiagramm zum Modul *camera_hardware.py* 54
5.11 Klassendiagramm zum Modul *camera_light_hardware.py* 55
5.12 Die GUI nach dem Start 56
5.13 Die Menüleiste der GUI 57
5.14 Das Eingabefenster *Process Sawn Wafer* 58
5.15 Das Eingabefenster *Load New Wafermap Hand* 59
5.16 Der *Automation Screen* in der GUI 60
5.17 Der *Measurement Screen* in der GUI 61
5.18 Der *Camera Screen* in der GUI 62
5.19 Der *Plot Screen* in der GUI 62
5.20 Registerkarte Wafermap in der GUI 63
5.21 Automatisch generierte Wafermap zur Reticleauswahl 64
5.22 Automatisch generierte Reticlemap zur Auswahl der DUT 65
5.23 Softwarestruktur der GUI Teil 1 67
5.24 Softwarestruktur der GUI Teil 2 67
5.25 Softwarestruktur der GUI Teil 3 68

6.1 Exemplarische Bilder aus dem Trainingsdatensatz für vereinzelte DUT Teil 1 74
6.2 Exemplarische Bilder aus dem Trainingsdatensatz für vereinzelte DUT Teil 2 75
6.3 Exemplarische Bilder aus dem Trainingsdatensatz für Wafer Teil 1 76
6.4 Exemplarische Bilder aus dem Trainingsdatensatz für Wafer Teil 2 77

6.5 Exemplarisches Labeling eines Bildes von einem Wafer 78
6.6 Automatische Datenerweiterung an einem Beispielbild 79
6.7 Trainingsüberwachung mit Tensorboard 82
6.8 Klassifikationsbewertung, modifiziert aus [9] 84
6.9 Visuelle Darstellung der IoU, modifiziert aus [15] 85
6.10 Ergebnis der Evaluierung zur Objekterkennung von vereinzelten DUT . . 86
6.11 Anwendung der Objekterkennung für vereinzelte DUT auf Bilder im Evaluierungsdatensatz 88
6.12 Ergebnis der Evaluierung zur Objekterkennung auf dem Wafer 89
6.13 Anwendung der Objekterkennung für DUT auf dem Wafer auf Bilder im Evaluierungsdatensatz Teil 1 90
6.14 Anwendung der Objekterkennung für DUT auf dem Wafer auf Bilder im Evaluierungsdatensatz Teil 2 92
6.15 Positionsanfahrt des Chucks mit Labeling durch trainiertes neuronales Netz 93
6.16 Ermitteln der Mittelpunkte der Objektrahmen 94
6.17 Beendigung der Korrekturfahrt nach Labeling des trainierten neuronalen Netzes 94
6.18 Objektfindung bei vereinzelten DUT 95
6.19 Nächstes zu kontaktierendes DUT 96

A.1 Lastenheft der Automatisierung des Darkboxprobers 108

B.1 Pflichtenheft der Automatisierung des Darkboxprobers 109

C.1 Aufgabenplan der Automatisierung des Darkboxprobers 110

E.1 Einstellung der Objekterkennung auf dem Wafer 114
E.2 Einstellen der Objekterkennung beim MPW mit unterschiedlichen Systemgrößen durch Erhöhung der Beleuchtungsstärke 115
E.3 Unterschiedliche Hardwarekonfiguration bei Anwendung der Objekterkennung auf einem Wafer Beispiel 1 116
E.4 Unterschiedliche Hardwarekonfiguration bei Anwendung der Objekterkennung auf einem Wafer Beispiel 2 117
E.5 Einstellen der Objekterkennung bei vereinzelten DUT 118

Tabellenverzeichnis

5.1 Relevante VISA-Kommandos für den SPA, sinngemäß aus [34] 41
5.2 Relevante VISA-Kommandos für den Chuck, sinngemäß aus [28] 47

6.1 Festgelegte Klassifikationen für die Objekterkennung bei vereinzelten DUT 70
6.2 Festgelegte Klassifikationen für die Objekterkennung bei Wafern 71
6.3 Grenzwerte der Objektgrößenklassifikationen, inhaltlich entnommen aus dem Modul *cocoeval.py* aus [35] 87

D.1 Ordner im Verzeichnis Software 111
D.2 Module im Verzeichnis Software zur Steuerung der Hardware 111
D.3 Module im Verzeichnis Software zur GUI 112
D.4 Module im Verzeichnis Software für die einzelnen Applikationen 113

Abkürzungen

AOI Automated Optically Inspection

API Application Programmable Interface

CNN Convolutional Neural Network

COCO Common Objects in Context

CSV Comma-sepearted Values

DUT Device Under Test

FC Fully Connected

FN False Negative

FP False Positive

GPIB General Purpose Interface Bus

GUI Graphical User Interface

HAW Hochschule für Angewandte Wissenschaften

IDE Integrated Development Environment

IoU Itersection over Union

KI Künstliche Intelligenz

MPW Multi Project Wafer

PCM Process Control Modul

R-CNN Region-based Convolutional Neural Network

RoI Region of Interest

RPN Region Proposal Network

SCPI Standard Commands for Programmable Instruments

SPA Semiconductor Parameter Analyzer

SPW Single Project Wafer

TN True Negative

TP True Positive

UML Unified Modeling Language

USB Universial Serial Bus

VISA Virtual Instrument Software Architecture

XML Extensible Markup Language

Quelltextverzeichnis

5.1 Implementation des Python-Paketes pyvisa, modifiziert aus [18] 40
5.2 Funktion zum Öffnen des Workspaces von EasyEXPERT über VISA . . . 40
5.3 VISA-Adresse des Chucks 46
5.4 Antwort des Chucks auf das VISA-Kommando **IDN?* 46
5.5 Umsetzung der Kommandos zur Chuckkommunikation in Methoden 47

6.1 Kommando zum Ändern der Bildgröße aller Trainingsdaten 78
6.2 Kommando zum Zusammenfassen der XML-Dateien in Trainings- und Testdaten im CSV-Format 80
6.3 Kommando zur automatischen Generierung des *.record*-Formates der Trainings- und Testdaten 81
6.4 Kommando zum Starten des Trainings zur Anwendung *Wafer* 81
6.5 Kommando zum Starten von Tensorboard zur Anwendung *Wafer* 82
6.6 Kommando zum Umsetzen des Trainingsergebnisses in ein Model zur Anwendung *Wafer* 83
6.7 Kommando zum Anwenden der Evalierung zur Anwendung *Wafer* 86

1 Einleitung

„Jeder Kunde kann seinen Wagen beliebig anstreichen lassen, wenn der Wagen nur schwarz ist“ [17, Henry Ford].

Mit diesem historischen Satz zeigte Henry Ford, einer der bekanntesten Menschen in der Automatisierungstechnik, am Anfang des 20. Jahrhunderts, dass Automatisierung ein entscheidender Faktor in der Wettbewerbsfähigkeit des Unternehmens ist. Heute ist die Automatisierungstechnik eine der Grundsäulen der Unternehmen, um vor allem bei den schnellen Innovationen und Weiterentwicklungen mitzuhalten. Henry Ford wollte schon damals mit seinem Satz ausdrücken, dass Automatisierungssysteme Grenzen aufweisen. 1903 waren diese Grenzen für den*die Endkunden*innen offensichtlich, wie beispielsweise die Farbe des Fahrzeuges. Heutzutage sind für den Endkunden keine Grenzen mehr erkennbar. Ein tieferer Blick in die Automatisierungssysteme zeigt jedoch, dass Grenzen vorhanden sind, wie beispielsweise die intuitiven Entscheidungen in einem Automatisierungsablauf, die in der Industrie häufig durch Menschen getroffen werden. Das Thema künstliche Intelligenz ist der neuste Ansatz, auch diese Hürde der Automatisierung zu überwinden.

In dieser Masterarbeit wird ein Automatisierungssystem entwickelt, in dem ein KI[1]-Algorithmus zur Objekterkennung integriert wird. Genauer gesagt wird ein System zur automatisierten Messung von Schutzbauteilen auf einem Wafer entwickelt. Damit jedoch nicht nur Wafer mit über 138.000 Bauteilen[2] gemessen werden können, sondern auch Bauteile die freiliegend aneinander gereiht sind, ist ein Bildverarbeitungsalgorithmus zur Positionsbestimmung der Bauteile notwendig. Normalerweise würde an dieser Stelle mit den bekannten Bildverarbeitungsalgorithmen ein Referenzbild des Bauteils genutzt werden, um dieses wiederzufinden. Da diese Masterarbeit jedoch in der Entwicklungsabteilung von Nexperia Germany GmbH durchgeführt wird, in der die optischen

[1] **K**ünstliche **I**ntelligenz

[2] siehe exemplarische Wafermap auf der DVD zur Masterarbeit (Masterthesis:/97_Dokumentation/04_Wafermap/pqc08.dpl)

Eigenschaften der Bauteile variieren, wird ein Bildverarbeitungssystem über ein Referenzbild keine ausreichende Genauigkeit darstellen. Daher wird in dieser Masterarbeit eine künstliche Intelligenz zur optischen Erkennung dieser Bauteile trainiert und in die Automatisierung eingebunden. Der Algorithmus trifft zukünftig die intuitive Entscheidung, an welcher Position sich ein Bauteil befindet. Warum es sich hierbei um eine eher intuitive Entscheidung handelt, wird im Kapitel zur künstlichen Intelligenz erläutert. Das Ziel dieser Masterarbeit ist nach Forderung des Lastenheftes, eine Genauigkeit von 70 Prozent bei der optischen Erkennung dieser Bauteile zu erreichen.

Diese Masterarbeit mit dem Titel *Prototypische Softwareentwicklung eines Automatisierungssystems unter Einbindung eines neuronalen Netzes zur Objekterkennung* gliedert sich in acht Kapitel. Im ersten Schritt werden alle notwendigen Grundlagen erläutert. Hierbei wird das Thema Halbleiterelektronik nur grob erklärt, da der Fokus dieser Arbeit auf der Automatisierung liegt. Neben der Beschreibung der verwendeten Hardware befinden sich in diesem Kapitel die notwendigen Informationen zu den verwendeten Modulen in Python. Abschnitt 2.4 verdeutlicht die Grundlagen zum Thema neuronale Netze. Nachdem im folgenden Kapitel die Aufgabenstellung vorgestellt wird, ist im Anschluss das Projektkonzept zu finden. Das Projekt als Ganzes wird nach dem Vorgehensmodell Rapid Prototyping [26] umgesetzt. Die Entwicklung des Quelltextes ist im Kapitel Implementierung erläutert. Es wird auf die Softwarestruktur eingegangen und die einzelnen entwickelten Module der Software werden vorgestellt. Außerdem beinhaltet dieses Kapitel die Vorstellung der entwickelten Bedieneroberfläche. Das Einbinden der *Tensorflow Object Detection API* ist in einem weiteren Kapitel beschrieben und in diesem Zusammenhang wird das Vorgehen zum Trainieren dieses Netzes erläutert. Die Evaluation des neuronalen Netzes nach dem Training ist ebenfalls Bestandteil dieses Kapitels. Außerdem wird die Integration der verwendeten *Tensorflow Object Detection API*[3] in die Gesamtanwendung vorgestellt.

Im Kapitel 7 wird der zugehörige Testplan zur Software vorgestellt. Neben dem Testplan wird der durchgeführte Langzeit- und Stabilitätstest beschrieben. Kapitel 8 und Kapitel 9 enthalten abschließend alle Erkenntnisse und mögliche Erweiterungen die während der Erstellung dieser Masterarbeit gesammelt werden konnten.

[3] **A**pplication **P**rogrammable **I**nterface, englisch für programmierbare Schnittstelle zur Anwendung

2 Grundlagen

Im Folgenden sind die wichtigsten Begriffe der Halbleiterelektronik erläutert, welche als Grundlage für das Verständnis dieser Masterthesis notwendig sind. Außerdem werden in diesem Kapitel die verwendeten Hardwarekomponenten vorgestellt. Nach der Beschreibung der Grundlagen zur eingesetzten Software in dieser Masterthesis, beendet die Erläuterung der Grundlagen von neuronalen Netzen in der Objekterkennung dieses Kapitel.

2.1 Halbleiterelektronik

Unter Halbleiterelektronik versteht man kleinstelektronische Bauelemente, die sich in Form und Funktion unterscheiden. Hierzu zählen nach [27] zum Beispiel Dioden und Transistoren.

2.1.1 DUT

Ein kleinstelektronisches System, wie zum Beispiel eine Diode, wird im Folgenden als DUT[1] bezeichnet. Die Erkennung solcher Systeme wird in dieser Masterthesis ausschließlich mit Dioden erprobt, wodurch auch der Begriff DUT in dieser Masterthesis mit einer Diode gleichgesetzt werden kann und stets ein System mit zwei Kontaktpads beschreibt. Die elektrotechnische Funktionalität eines DUT findet in dieser Masterthesis keine Relevanz, da hier nur die optische Erkennung solcher DUT thematisiert wird.

[1] **D**evice(s) **U**nder **T**est, englisch für Gerät im Test

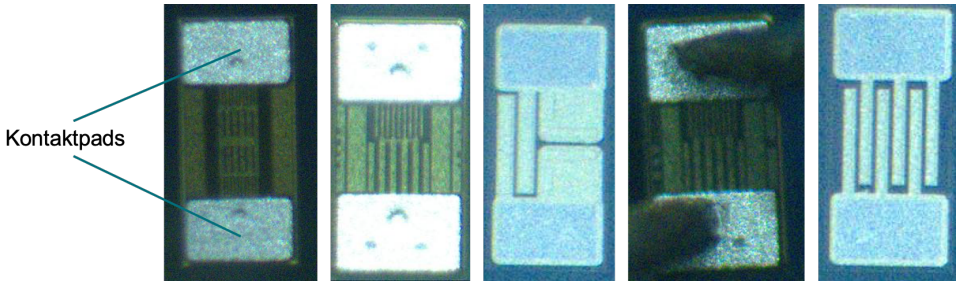

Abbildung 2.1: Beispiele für DUT

In Abbildung 2.1 sind verschiedene Bauteile zu erkennen. Offensichtlich ist bei diesen Bauteilen, dass sie zwei Anschlusspads besitzen. Des Weiteren befinden sich zwischen den Pads die Systemstrukturen, die für die Funktionalität des DUT verantwortlich sind. Durch die Variation dieser Systemstrukturen generiert die Entwicklungsabteilung von Nexperia Germany GmbH Neuentwicklungen. Da diese Masterthesis in dieser Abteilung umgesetzt wird, ist im weiteren Verlauf davon auszugehen, dass sich diese Systemstrukturen optisch ständig verändern. Daher sind die ersten Indikatoren zur Erkennung eines solchen DUT die zwei Anschlusspads, die in der Regel eine ähnliche Größe und Form aufweisen. Ein weiterer Indikator ist die Systemstruktur zwischen den Pads, welche beliebig komplex werden kann, jedoch ähnliche Wiederholungen zu vorherigen Systemstrukturen beinhaltet.

Um ein solches DUT zu produzieren, werden verschieden aufwendige Fertigungsverfahren auf eine Siliziumscheibe angewendet. Hierbei werden Schichten geätzt, belichtet und neue Materialien aufgetragen. Dieser gesamte Prozess wird mehrfach wiederholt. Für detailliertere Information zu Halbleitern wird auf [21] und [27] verwiesen.

2.1.2 Reticle

Für die schnellere Durchführung dieser Fertigungsschritte werden diese auf ein sogenanntes Belichtungsfeld angewendet, welches nach [16] auch unter der Bezeichnung Reticle bekannt ist. Im Folgenden wird für ein Belichtungsfeld immer der Begriff Reticle verwendet.

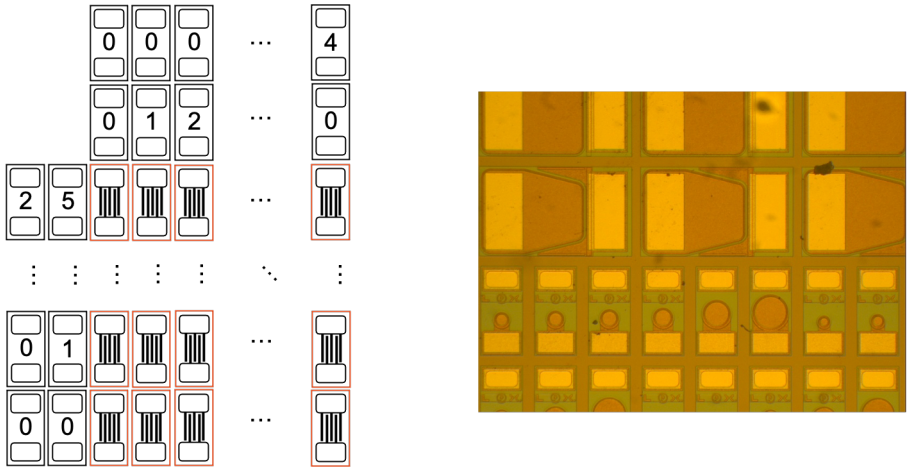

Abbildung 2.2: Aufbau eines Reticles

In Abbildung 2.2 ist auf der linken Seite ein Beispiel der Organisation eines Reticles aufgezeigt. Dort sind mehrere DUT und Nummerierungen angeordnet. Die Nummerierungen, und die DUT verfügen über Kontaktpads, wobei die Nummerierungen keinerlei Funktion besitzen. Diese sind nur zur Nummerierung der DUT im Reticle gedacht. Zu verstehen ist das Reticle wie eine Matrix, die ihren Ursprung in den meisten Fällen links unten hat. Um ein DUT im Reticle zu finden, benötigt man somit lediglich die X-Y-Koordinate und kann über die Nummerierung die DUT zuordnen. Das Beispiel zeigt ein Reticle mit 40 Spalten horizontal und 25 Zeilen vertikal. Im Reticle befinden sich in diesem Beispiel somit 1000 verschiedene DUT. In Abbildung 2.2 ist auf der rechten Seite zu erkennen, dass die Abmaße der DUT im Reticle unterschiedlich sein können. Betrachtet man die größeren DUT im oberen Bereich des Bildes, fällt auf, dass diese eine andere Orientierung mit den Pads im Vergleich zu den kleineren DUT im unteren Bereich des Reticles haben.

Zusammengefasst ist ein Reticle für die Produktion ein wichtiger Bestandteil in der Halbleiterbearbeitung, da hiermit mehrere Systeme gleichzeitig prozessiert werden können. Des Weiteren bietet dieses bei einem fertig prozessierten Wafer, eine optische Orientierungsmöglichkeit durch die Nummerierung. Durch die Möglichkeit, die Zeilen oder Spalten im Reticle unterschiedlich breit zu gestalten, kann die Vielfalt der DUT vergrößert werden. In einem Entwicklungsprozess kann der*die Entwickler*in somit mehrere Systemstrukturen und verschiedene Systemgrößen in einem Reticle parallel prozessieren lassen. Dem*der Entwickler*in wird somit die Option geboten, verschiedene Variationen in einem Entwicklungszyklus eines Produktes zu testen. Die Nummerierungen aus Ab-

bildung 2.2 finden vor allem auf dem MPW[2] Anwendung, da hier DUT mit ihren Strukturen wiedergefunden werden müssen. Auf einem SPW[3] findet man ebenfalls Reticles, diese haben jedoch keine Nummerierung, da alle DUT die gleichen Systemstrukturen beinhalten.

2.1.3 Wafer

Ein Wafer lässt sich mit der folgenden Beschreibung sinngemäß nach [21] zusammenfassen. Ein Wafer ist eine sehr dünne runde Scheibe, welche aus einem Halbleitermaterial besteht. Auf dieser Scheibe können über verschiedene Fertigungsschritte, wie beispielsweise das Ätzen von Strukturen, verschiedene mikroelektronische Systeme realisiert werden.

In Abbildung 2.3 ist ein 8-Zoll[4] Wafer zu sehen. Diese Wafer sind mit einer sogenannten Notch gekennzeichnet. Nach [16] ist die Notch am untersten Punkt des Wafers als kleine Ecke zu erkennen. Diese wird in der Fertigung zur Arretierung des Wafers verwendet. Bei anderen Wafergrößen wird die Ausrichtung der Scheibe über andere markante Kennzeichnungen am Wafer realisiert. Diese Masterthesis behandelt jedoch thematisch nur die 8-Zoll Wafer mit einer Notch.

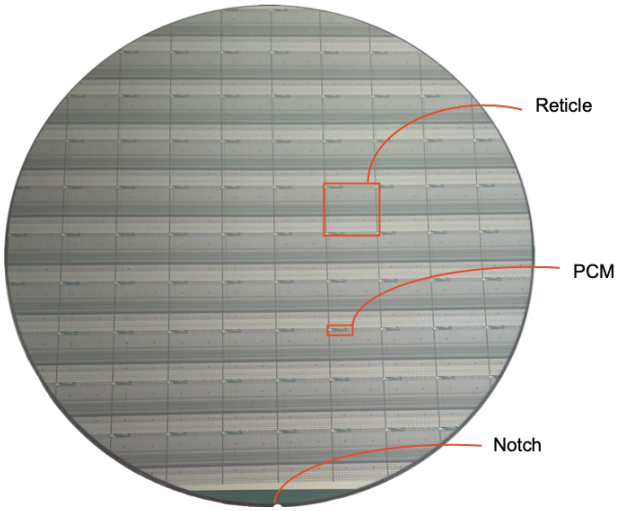

Abbildung 2.3: Exemplarischer 8-Zoll Wafer mit Notch

[2] **M**ulti **P**roject **W**afer, englisch für Wafer mit mehreren Systemprojekten
[3] **S**ingle **P**roject **W**afer, englisch für Wafer mit einem Systemprojekt
[4] 8-Zoll sind $203,2mm$

Über den Waferdurchmesser von 203,2mm werden mehrere Reticles zur Produktion der mikroelektronischen Systeme auf dem Wafer angeordnet. Die einzelnen Reticles sind als Gitterstruktur in Abbildung 2.3 zu erkennen und können willkürlich auf dem Wafer angeordnet sein, wobei in der Produktion diese Anordnung häufig von der Ausbeute an realisierten Systemen auf dem Wafer abhängig gemacht wird. Auf einem Wafer können bei einem DUT mit den Maßen 0,6mm x 0,3mm circa 138.000 DUT produziert werden. Eine weitere wichtige Größe zum Wafer ist der Pitch. Dieser definiert nach [4] den Abstand zwischen den DUT untereinander und setzt sich aus den Abmaßen des DUT und einer aufsummierten Sägebahnbreite zusammen. So ist bei einem DUT-Maß von 0,6mm x 0,3mm beispielsweise ein Pitch von 0,63mm x 0,33mm auf dem Wafer vorhanden. Außerdem wird zu jedem Wafer neben dem DUT- und ein Reticle-Pitch angegeben. In Abbildung 2.3 ist außerdem ein PCM[5]-Feld gekennzeichnet, mit dem nach [16] wichtige Prozessschritte kontrolliert werden können. Dabei werden zum Beispiel die notwendigen Widerstandswerte der Kontakte im Nachgang über das PCM geprüft. In jedem Reticle auf dem Wafer ist ein PCM zu finden. Unterschiedliche Prozessschritte fordern verschiedene Kontrollschritte, wodurch jedes PCM-Modul auf das zu prozessierende System angepasst ist.

Gesägte Wafer

Während des Sägeprozesses wird der Wafer entlang der Sägebahnen zersägt, wodurch die DUT voneinander getrennt werden[6]. Dabei wird der zersägte Wafer auf eine Folie geklebt, sodass die einzelnen DUT weiterhin in der bekannten Anordnung zu finden sind.

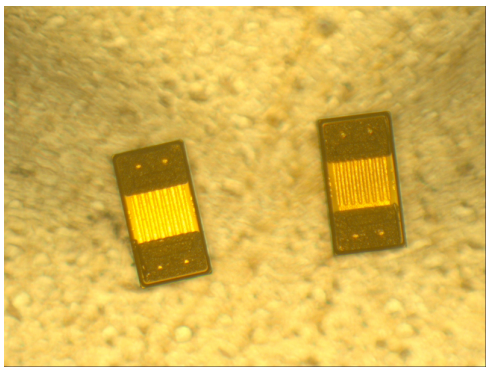

Abbildung 2.4: Exemplarisches Bild vereinzelter DUT

[5] **P**rocess **C**ontrol **M**odule, englisch für Prozesskontrollmodul
[6] Einige Grundlagen zum Zersägen von Wafern sind in [4] zu finden.

Abgelöst von der Folie und voneinander getrennt angeordnet, sehen die DUT auf einer separaten Folie wie in Abbildung 2.4 aus. Vor allem für die Messung von bauteilspezifischen Kenngrößen, wie zum Beispiel die Kapazität, ist diese Anordnung der DUT wichtig. Durch das Vereinzeln der DUT wird der gegenseitige Einfluss der Bauteile drastisch reduziert. Im Vergleich ist das Messergebnis der Kapazität in diesem Messaufbau deutlich präziser als die Kapazitätsmessung von DUT auf dem Wafer.

Organisation des Wafers

Als Entwicklungsbestandteil wird zu jedem Wafer eine Wafermap erstellt, in der verschiedene Informationen über den Wafer abgelegt sind. In der Kopfzeile befinden sich Informationen zu dem Abstand zwischen den einzelnen DUT, die Anzahl der Gutsysteme und die Anzahl der Gesamtsysteme auf dem Wafer. Eine Norm für die einheitliche Nummerierung von Systemen auf dem Wafer existiert nicht, jedoch wird bevorzugt das folgende Koordinatensystem auf einem Wafer verwendet.

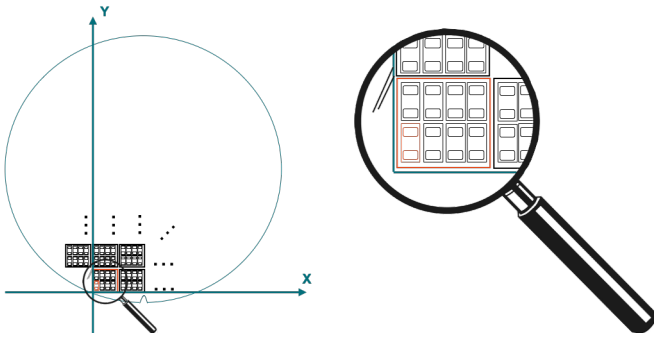

Abbildung 2.5: Koordinatensystem auf dem Wafer

In Abbildung 2.5 ist zu erkennen, dass das gesamte Koordinatensystem auf dem Wafer an dem linken DUT in der untersten Zeile orientiert ist. Auch die Reticles orientieren sich an diesem DUT, somit ist das DUT 0,0 auch im Reticle 0,0. Die X-Achse ist im Koordinatensystem sowohl positiv als auch negativ über den Wafer vertreten, die Y-Achse nur mit positiven Werten. In Abbildung 2.5 ist in orange zum Einen das DUT 0,0 und zum Anderen das Reticle 0,0 markiert. In der Lupe ist dieser Bereich vergrößert, wobei das Reticle 0,0 und das DUT 0,0 wieder in orange zu sehen sind.

Single Project Wafer - SPW

Auf einem SPW wird nur ein System mehrfach realisiert. Auf diesem Wafer findet man keine weiteren Systeme, als das in der Wafermap bezeichnete System. Auf einem 8-Zoll Wafer mit den Systemabmaßen 0.63x 0.33mm befinden sich über 138.000 gleiche Systeme. Der SPW dient zur Massenproduktion.

Multi Project Wafer - MPW

Im Gegensatz zum SPW werden auf einem MPW mehrere Systeme in einem Reticle realisiert, welches sich über den Wafer verteilt wiederholt. MPW werden häufig in der Entwicklung verwendet, damit mehrere Entwicklungspfade gleichzeitig auf einem prozessierten Wafer realisiert werden können. Diese Verfahrensweise ermöglicht es, schnellere Entwicklungserfolge umzusetzen. Ähnlich wie die Organisation eines Arrays, werden die Nummerierungen des Reticles durchgeführt. Über eine Positionsangabe im Reticle zu einem DUT kann dieses einer spezifischen Systembeschreibung zugeordnet werden. Die Nummerierung dieses Reticles unterscheidet sich unter den Entwicklern. Daher ist in Abbildung 2.2 lediglich ein Beispiel von vielen möglichen Organisationen des Reticles zu sehen.

2.2 Verfügbare Hardware

Für die messtechnische Untersuchung der DUT auf einem Wafer wird eine präzise Hardware benötigt, mit der die Positionierung der Messspitzen auf den kleinen Pads der DUT möglich ist. In diesem Abschnitt wird das bereits von Nexperia Germany GmbH festgelegte Hardwarekonzept vorgestellt, mit dem der softwaretechnische Prototyp umgesetzt werden soll.

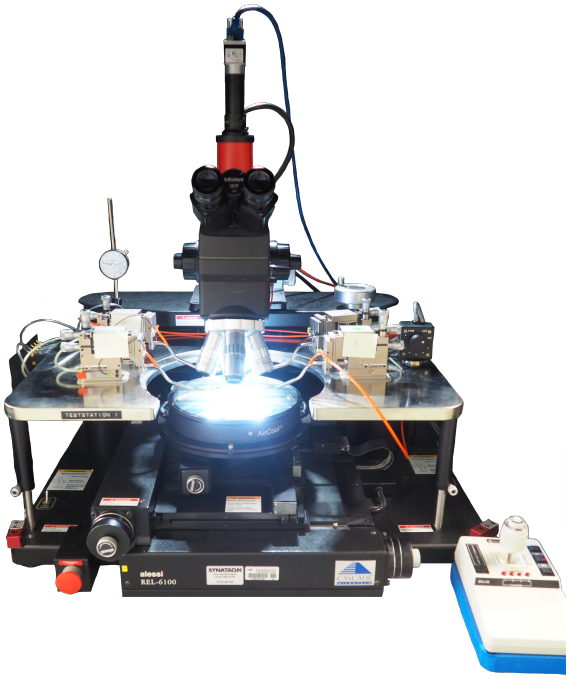

Abbildung 2.6: Hardwareaufbau des Waferprobers in der Darkbox

Ein*e Messingenieur*in kann mit Hilfe eines sogenannten Waferprobers die Pads eines DUT manuell unter den Messspitzen eines Messgerätes positionieren. Dabei kann das DUT je nach Messaufgabe freiliegend sein oder sich noch auf dem ungesägten Wafer befinden. Eine präzise Positionierung dieser Messspitzen für die Messaufgabe ist dabei möglich, da der*die Messingenieur*in die DUT mit einer optischen Einheit extrem vergrößert betrachten kann. Im Hardwareaufbau in Abbildung 2.6 ist der Wafer in dem stark beleuchteten Bereich, mittig vom Bild, zu finden. Die einzelnen Messungen müssen aufgrund der hohen Lichtempfindlichkeit der DUT in einem dunklen Raum durchgeführt werden. Nachdem die Messspitzen kontaktiert sind, schließt der*die Messingenieur*in den Deckel der Darkbox[7], in der sich der Waferprober aus Abbildung 2.6 befindet. Im Folgenden wird der Begriff Waferprober für die oben erkennbare Hardware in Abbildung 2.6 verwendet. Mit dem Begriff Darkboxprober wird im Folgenden der gesamte Aufbau der Darkbox inklusive der zugehörigen Messinstrumente bezeichnet.

Der Darkboxprober besteht aus einer Marmorplatte, auf der ein Chuck[8] gestellt ist. Dieser kann den Wafer in der X-, Y-, Z- und rotativen Achse bewegen. Des Weiteren beinhal-

[7] englisch für eine abgedunkelte Box
[8] englisch für Spannvorrichtung

tet der Darkboxprober eine Tischebene, auf der Messspitzen befestigt werden können. Mit der bereits erwähnten optischen Einheit wird ein Bereich auf dem Wafer vergrößert dargestellt. Auf dieser Einheit ist eine Kamera installiert, die parallel zur direkten Linse das Bild aus der optischen Einheit einliest. Mit dieser Kamera sind nach [3] Bilder in einer Auflösung von 1920x1200 Pixel möglich. Die Beleuchtungstechnik des Wafers ist ein wichtiger Bestandteil des Darkboxprobers. Hierbei ist eine Koaxial-Beleuchtung und ein Direktlicht[9] realisiert, welche getrennt voneinander angesteuert werden können.

2.2.1 Chuck

Der Chuck bietet dem*der Nutzer*in zwei Möglichkeiten zum Verfahren. Zum Einen kann die gewünschte Position des Wafers über den Joystick unten rechts in Abbildung 2.6 händisch angefahren werden. Hierbei lässt sich die Geschwindigkeit des Chucks von dem*der Nutzer*in steuern. Eine weitere Möglichkeit zum Verfahren bietet die VISA[10]-Schnittstelle.

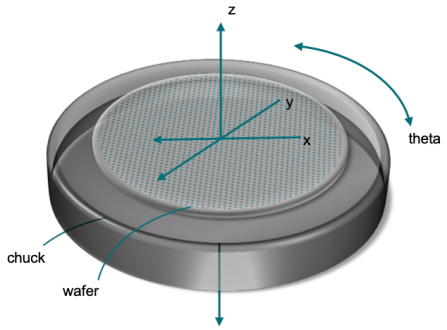

Abbildung 2.7: Koordinatensystem des Chucks

Über diese Schnittstelle sind Kommandos zur Vorgabe der nächsten Position in der X-, Y-, Z- und rotativen Achse des Chucks möglich. Außerdem können relative Positionsdaten zur aktuellen Position des Chuck vorgegeben werden. Die interne Regelung des Chucks setzt bei diesen Kommandos eine ausreichende Genauigkeit zur Kontaktierung von Messspitzen auf den Pads eines DUT um. Der Systemmittelpunkt des Chucks wird als sogenannte Homeposition[11] bezeichnet. An diesem Punkt befindet sich der Koordinatenursprung für

[9] Für detailliertere Informationen zur Beleuchtungstechnik für Bildverarbeitungsaufgaben, zu denen die Koaxial-Beleuchtung und das Direktlicht zählen, sei auf [11] verwiesen.
[10] **V**irtual **I**nstrument **S**oftware **A**rchitecture, englisch für Virtuelle Instrumenten Softwarearchitektur
[11] englsich für Startposition

die X-, Y- und Z-Achse. Der Bewegungsbereich in der X- und Y-Achse ist über einen Radius von 103mm zum Systemmittelpunkt begrenzt. Außerdem ist der Chuck in der Z-Achse auf einen Bewegungsbereich von +-50mm zur Homeposition begrenzt. Der rotative Bewegungsbereich liegt zwischen $-8°$ und $+8°$. Für weitere Informationen sei auf das Handbuch zur Hardware in [7] hingewiesen.

2.2.2 Semiconductor Parameter Analyzer

Zur funktionalen Validierung werden die verschiedenen DUT über ein großes Spannungs- und Stromspektrum gemessen. Hierfür ist im Hardwareaufbau der SPA[12] als Messgerät integriert. In Abbildung 2.6 sind auf der zweiten Tischebene Messspitzen zu erkennen, die an dem SPA angeschlossen sind.

Abbildung 2.8: Semiconductor Parameter Analyzer von Keysight, übernommen aus [34]

In Abbildung 2.8 ist eines der Messgeräte zum Darkboxprober abgebildet. Auf dem SPA befindet sich die Software EasyEXPERT, die als grafische Schnittstelle zur Messgerätesteuerung dient. Über die Software können eigene Messskripte erstellt und abgelegt werden, wodurch sich bereits viel messtechnisches Know-How von Nexperia Germany GmbH auf dem SPA befindet. Verschiedene Messskripte zur strom- oder spannungsgetriebenen Messung sind vorkonfiguriert und werden im Arbeitsalltag häufig zur Validierung von DUT wiederverwendet. Im EasyEXPERT ist ein Workspace[13] eingerichtet, unter dem verschiedene *Preset Groups*[14] zu finden sind. In diesen *Preset Groups* sind die einzelnen Messskripte kategorisiert. Über den Touchscreen kann das Messgerät bedient werden. Des

[12] **S**emiconductor **P**arameter **A**nalyzer, englisch für Halbleiter-Parameteranalysator
[13] englisch für Arbeitsplatz
[14] englisch für voreingestellte Gruppen

Weiteren kann der SPA über zwei verschiedene Wege mittels VISA-Protokoll gesteuert werden. Zum Einen bietet der SPA die Möglichkeit, direkte Messbefehle über VISA entgegenzunehmen. Zum Anderen kann auch der EasyEXPERT über VISA gesteuert werden, sodass das bereits existierende messtechnische Know-How auf dem Messgerät direkt verwendet werden kann. Diese VISA-Kommunikation kann in diesem Fall nur über Ethernet erfolgen. Der technische Aufbau für diese Handhabung des Messgerätes ist in [34] zu finden.

Messspitzen

Die Messspitzen auf der zweiten Tischebene des Darkboxprobers können in der X-, Y- und Z-Ebene über Justierschrauben eingestellt werden. Vor der Messung muss der*die Messingenieur*in die Messspitzen auf dem DUT kontaktieren.

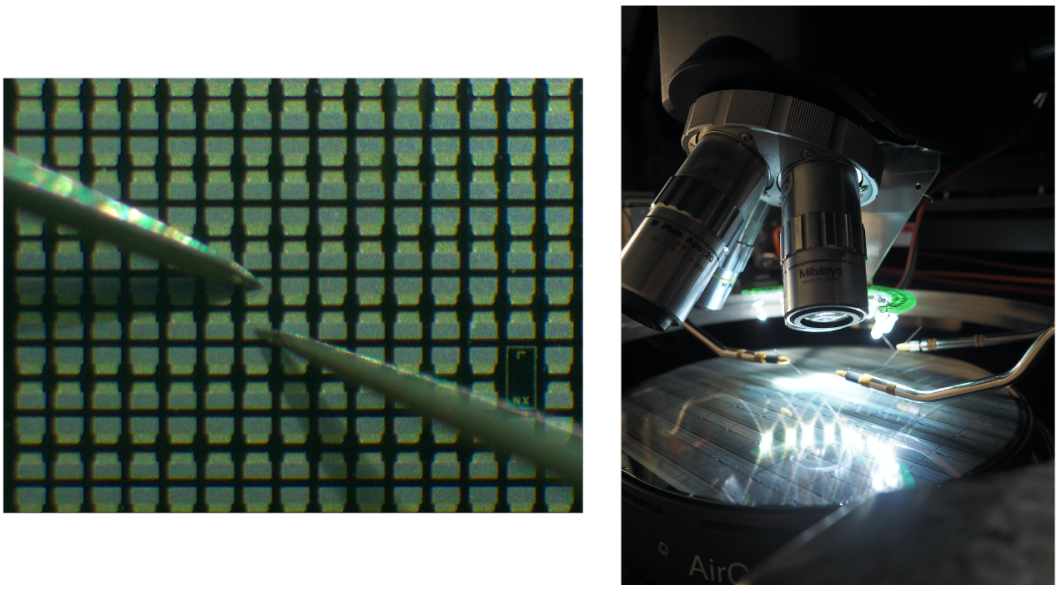

Abbildung 2.9: Messspitzen am Darkboxprober

In Abbildung 2.9 ist auf der rechten Seite ein Wafer zu erkennen, auf dem die Messspitzen kontaktiert sind. Ein Blick durch die Optik zeigt die vergrößerte Aufnahme, die in Abbildung 2.9 links zu sehen ist. Zur Kontaktierung werden die Messspitzen in der Bildmitte auf dem DUT kontaktiert. Hierbei ist vor allem wichtig, dass die Messspitzen fest auf den Wafer gepresst werden, um eine gute Kontaktierung zu realisieren. Oftmals wird die Messspitze etwas über die Kontaktfläche des DUT geschoben, um über das Kratzen auf

der Fläche die Kontaktierung zu verstärken. Die Messspitzen haben keine feste Position im Bild und können je nach Messaufgabe unterschiedlich angeordnet werden.

2.2.3 Beleuchtung

Im Darkboxprober gibt es zwei verschiedene Beleuchtungssysteme. In einem dieser Systeme wird eine externe Lichtquelle über einen Lichtschlauch in die Darkbox geführt. Diese realisiert eine Koaxial-Beleuchtung wie in Abbildung 2.10.

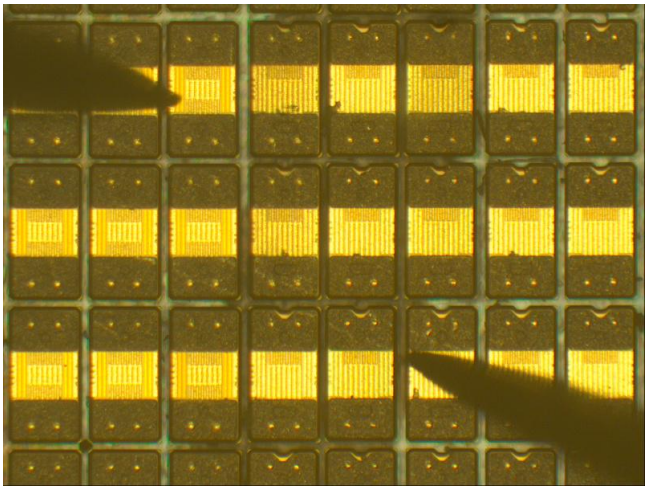

Abbildung 2.10: Exemplarischer Wafer unter der Koaxial-Beleuchtung

Bei dieser Lichtquelle ist zu erkennen, dass vor allem die geraden Flächen die Reflektion verursachen. Sobald Kanten oder abgerundete Flächen vorkommen, wie beispielsweise die Messspitzen in Abbildung 2.10, reflektieren diese das Licht in eine andere Richtung als die der Kamera, sinngemäß nach [11]. Die runden Flächen oder Kanten werden dann sehr dunkel dargestellt. Raue Oberflächen, wie beispielsweise die Kontaktpads, reflektieren das Licht nicht in die Richtung der Optik, wodurch diese dunkler als die glatten geraden Flächen sind. Durch die Spiegelung des Lichtes in Richtung der Optik erscheinen die glatten geraden Flächen golden und hell unter der Koaxial-Beleuchtung.

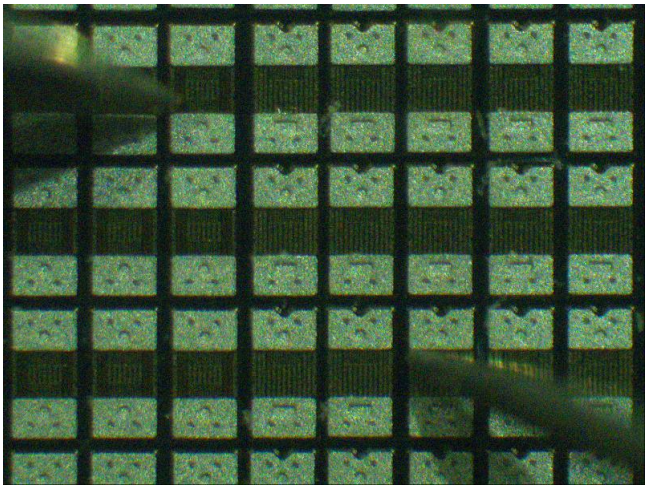

Abbildung 2.11: Exemplarischer Wafer unter LED Licht

Des Weiteren kann über einen LED-Halbkreis unter der Optik der Wafer beleuchtet werden. Hierbei ist der Wafer in der Originalfarbe zu erkennen, wobei ein Blaustich durch das LED-Licht vorhanden ist. Die Kontaktpads sind silbern und die Sägebahnen heben sich farblich deutlich von den Systemstrukturen im DUT ab.

2.2.4 Laborrechner

Das Hardwarekonzept zu dieser Masterthesis ist an der Anlage bereits vorhanden. Als Kernstück ist der Laborrechner zu sehen, auf dem die zukünftige prototypische Software ausgeführt wird.

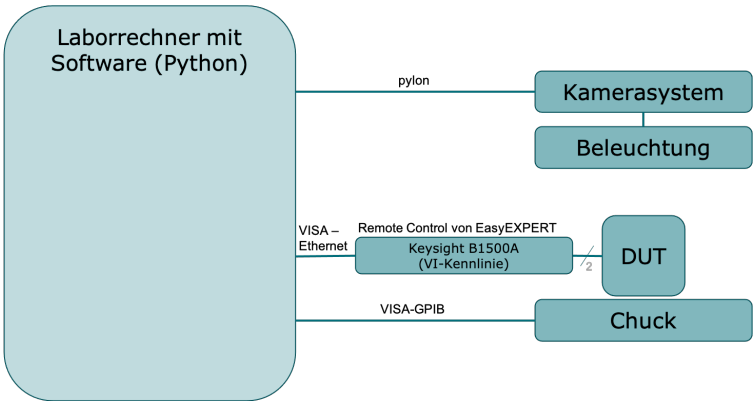

Abbildung 2.12: Hardwareaufbau mit Laborrechner

In Abbildung 2.12 ist das bereits vorhandene Hardwarekonzept der Anlage zu sehen. Der Chuck ist über GPIB[15] an den Laborrechner angeschlossen. Es soll das VISA-Protokoll, welches in Unterabschnitt 2.3.2 erläutert wird, zur Steuerung des Gerätes verwendet werden. Ebenfalls wird der SPA über das VISA-Protokoll gesteuert, jedoch wird hierbei eine Ethernet-Verbindung verwendet. Wie schon in Unterabschnitt 2.2.2 beschrieben, greifen die VISA-Kommandos nicht direkt auf die Messgerätesteuerung zu, sondern werden über die zwischengeschaltete Messgerätesoftware EasyEXPERT verarbeitet. EasyEXPERT übernimmt auf dem SPA somit die Handhabung des Messgerätes. Am Messgerät wird dann das DUT über die Messspitzen angeschlossen. Die Beleuchtung wird über eine serielle Schnittstelle realisiert und ist über USB[16] an dem Laborrechner angeschlossen. Die Kamera wird ebenfalls über USB mit dem Laborrechner verbunden.

2.3 Software

Die Software dieser Masterthesis wird mit der Sprache Python entwickelt. Im Folgenden werden die verwendeten Versionen der Module in Python vorgestellt.

2.3.1 Python

In dieser Masterthesis wird mit der Open-Source-Distribution Anaconda gearbeitet. Zum Zeitpunkt der Installation wurde die Python Version 3.7.6, 64-bit gestellt, mit der in der gesamten Masterthesis gearbeitet wird. Es wird die IDE[17] Spyder verwendet. Außerdem wird der Python-Code nach der PEP8-Richtlinie[18] entwickelt. Neben Anaconda müssen noch weitere Python-Packages installiert werden:

- OpenCV (Version 4.2), Informationsfindung unter [20]
- pyvisa (Version 1.10), Informationsfindung unter [6]
- pypdf2 (Version 1.26), Informationsfindung unter [30]
- pyplon (Version 1.5.4), Informationsfindung unter [1]
- PyQt (Version 5.14.2), Informationsfindung unter [25]

[15] **G**eneral **P**urpose **I**nterface **B**us, englisch für Allzweck-Schnittstellenbus
[16] **U**niversial **S**erial **B**us, englisch für universeller serieller Bus
[17] **I**ntegrated **D**evelopment **E**nvironment, englisch für integrierte Entwicklungsumgebung
[18] Richtlinie für einen einheitlichen Quelltext von Pylint

- pyserial (Version 3.4), Informationsfindung unter [24]
- tensorflow (Version 2.0.0), Informationsfindung unter [22]

2.3.2 Python-Paket zur Kommunikation - pyvisa

Das VISA-Protokoll lässt sich sinngemäß nach [18] in den folgenden Sätzen beschreiben. Unter der Nutzung des VISA-Protokolls stehen verschiedene Schnittstellen zur Verfügung. Das VISA-Protokoll verbindet viele existierende Protokollformate. Dabei wird ein Lese- oder Schreibbefehl auf das jeweils genutzte existierende System umgesetzt. Neben GPIB und Ethernet gibt es viele weitere Protokollformate, die mit dem VISA-Protokoll genutzt werden können. Ein Beispiel aus der VISA-Dokumentation ist nach [8] das SCPI[19] **IDN?*. Hierbei kann die Identifikationsnummer des Gerätes abgefragt werden. Weitere Informationen zu den einzelnen Befehlen im Protokoll sind unter [18] zu finden oder direkt beim Gerätehersteller des VISA-fähigen Gerätes. In dieser Masterthesis wird die Bibliothek pyvisa verwendet, in der die Kommandos für das Lesen und Schreiben über VISA bereits programmiert sind. Informationen hierzu wurden aus [6] entnommen.

2.3.3 Python-Paket zur Kamera - pypylon

Für die Anbindung der Kamera an die Programmiersprache Python wird das Schnittstellenpaket *pylon Camera Software Suite Interface* genutzt. Für detaillierte Informationen zu dieser Software sei auf [1] verwiesen.

Zur Steuerung der Baslerkamera wird daher das Open-Source-Projekt pypylon von Basler verwendet. Über die Anweisung *pip install pypylon* lässt sich die Modulinstallation automatisch durchführen. Die Bibliothek wird daraufhin automatisch installiert. Ein Beispiel zur Implementierung des Moduls findet man unter [2], welches angepasst in dieser Masterthesis verwendet wird.

[19] **S**tandard **C**ommands for **P**rogrammable **I**nstruments, englisch für standard Kommandos für programmierbare Instrumente, Informationen hierzu unter [8]

2.3.4 Python-Paket zur Benutzeroberfläche - PyQt

In dieser Masterthesis wird die Benutzeroberfläche mit der Python-Bindung PyQt5 umgesetzt. Hierzu kann nach [25] über den Befehl *pip install pyqt* die automatische Installation des Paketes gestartet werden. PyQt ist plattformunabhängig, jedoch wird in dieser Masterthesis nur auf einem System mit Windows 10 als Plattform gearbeitet. Ebenfalls wird der QtDesigner als Hilfsmittel verwendet, um die GUI[20] vorerst grafisch zu konstruieren. Das PyQt Paket bietet Funktionen für die Umsetzung des Dateiformates *ui* aus dem QtDesigner in Python, sodass die grafisch konstruierte GUI schnell in Python-Quelltext umgesetzt werden kann.

2.4 Neuronale Netze

Für die Grundlagen zur Anwendung von neuronalen Netzen in der Bildverarbeitung werden im Folgenden die Begriffe Perzeptron, Training, Klassifikation und Objekterkennung erläutert. Mit der Vorstellung eines speziellen Algorithmus zur Objekterkennung wird dieses Unterkapitel beendet.

Die folgende Erläuterung zu den Grundlagen der neuronalen Netze ist aus [10] sinngemäß entnommen. Unter einem neuronalen Netz versteht man die Verknüpfung von einzelnen Neuronen zu einem zusammenhängenden Netz. Der Grundstein für diese Idee wurde bereits 1958 von Frank Rosenblatt gelegt, der sich mit der mathematischen Beschreibung eines Neurons beschäftigt und dieses ausformuliert hat. Bekannt ist diese Beschreibung unter dem Namen Perzeptron.

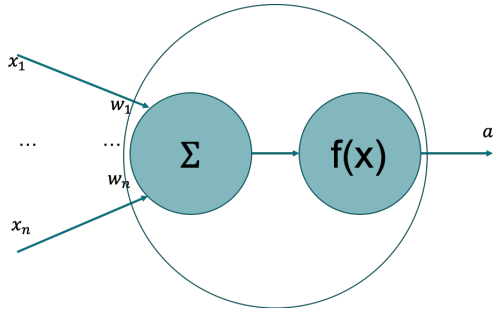

Abbildung 2.13: Das Perzeptron, modifiziert aus [29]

[20] **G**raphical **U**ser **I**nterface, englisch für grafische Benutzerschnittstelle

In Abbildung 2.13 ist die mathematische Beschreibung eines Perzeptrons zu erkennen. Diese ist angelehnt an die Funktionsweise eines biologischen Neurons, wie es beispielsweise im menschlichen Gehirn vorkommt. Mehrere Eingangssignale **x** werden jeweils mit einer Gewichtung **w** multipliziert und aufsummiert. Über diese Summe und eine Entscheidungsfunktion **f(x)** wird ein Ausgabewert **a** generiert. Die verwendete Aktivierungsfunktion **f(x)** im Perzeptron kann unterschiedliche Beschreibungen haben.

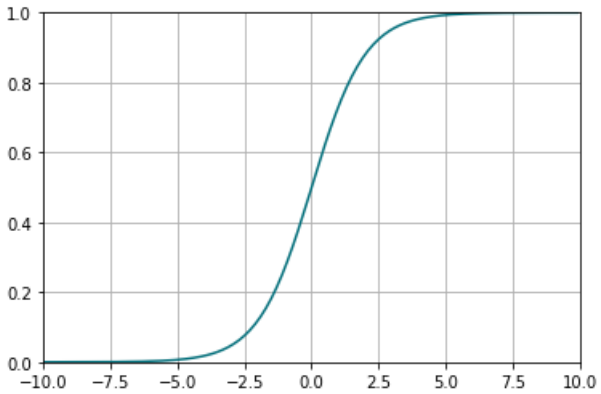

Abbildung 2.14: Die Sigmoid-Funktion (Python Plot)

Ein Beispiel für die Aktivierungsfunktion ist die Sigmoid-Funktion, wie sie in Abbildung 2.14 zu sehen ist.

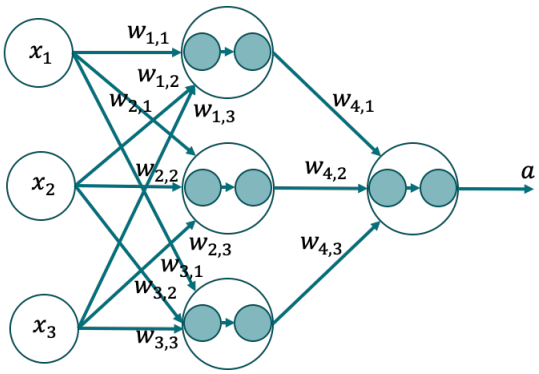

Abbildung 2.15: Exemplarisches neuronales Netz, modifiziert aus [29]

Exemplarisch ist ein neuronales Netz in Abbildung 2.15 zu erkennen. Die Struktur des Perzeptrons aus Abbildung 2.13 wiederholt sich in dem Netz mehrfach. Wie viele Perzeptrone in diesem Netz vorhanden sind und wie diese miteinander verknüpft sind, hängt

von der gewählten Architektur des Netzes ab. Eine weitere Variable ist die Tiefe des Netzes. Bei einer großen Netztiefe liegen mehr Spalten mit Neuronen zwischen dem Eingabe- und dem Ausgabevektor vor als bei einem Netz mit einer geringeren Tiefe. Um das Netz für eine bestimmte Aufgabe zu verwenden, muss es trainiert werden.

2.4.1 Training

Das Training eines neuronalen Netzes kann man sich, sinngemäß nach [23], wie den Lernprozess eines Menschens vorstellen. Als einfaches Beispiel hierfür wird das Lernen des Addierens verwendet. Einem Menschen oder in diesem Fall einem Kind würde man einige Beispiele von Aufgaben zeigen, wobei man zum Einen die beiden zu addierenden Zahlen aufzeigt und zum Anderen das Ergebnis. Diese Aufgabe würde man mit verschiedenen Werten wiederholen, wodurch ein Lernprozess beim Kind erreicht wird. In Analogie dazu kann auch das Training eines neuronalen Netzes gesehen werden. In der folgenden Grafik ist dieser Prozess modellbasiert aufgezeigt.

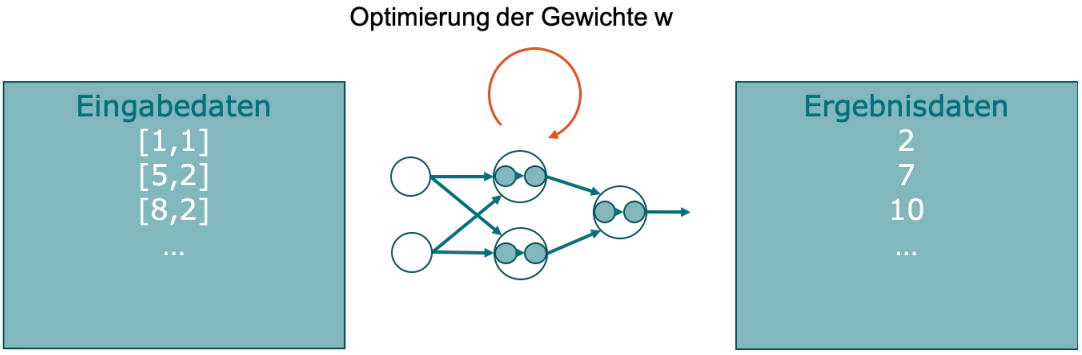

Abbildung 2.16: Der Aufbau eines Trainings von neuronalen Netzen

Dem neuronalen Netz werden in diesem Beispiel zwei Zahlen als Eingabedaten, die links in Abbildung 2.16 zu erkennen sind, gegeben. Zu diesen zwei Zahlen wird außerdem das Ergebnis, auf der rechten Seite in Abbildung 2.16, zugeordnet. Auf diese Aufgabe wird das Netz jetzt trainiert beziehungsweise optimiert, sodass bei Eingabe dieser Daten und den neuen optimierten Gewichten **w** das gewünschte Ergebnis vom neuronalen Netz ausgegeben wird. Bei einer größeren Datenmenge im Trainingsdatensatz werden folglich mehr Daten in den Optimierungsprozess eingebunden, wodurch das Netz eine sogenannte Transferleistung erlernen kann. Unter dieser Transferleistung versteht sich, dass das Netz über die eingeprägten Gewichtungen des Netzes eine ähnliche Aufgabe,

wie die bekannten Aufgaben aus dem Trainingsdatensatz, lösen kann. Das Ziel ist es, neue Aufgaben mit bisher unbekannten Ergebnissen zu lösen. Die Qualität des Training ist somit abhängig von der Menge der Daten im Datensatz. Je nach Aufgabenkomplexität muss somit der Testdatensatz, welcher aus Eingabe- und Ergebnisdaten besteht, vergrößert werden. Etwas komplexer zusammengefasst kann man sich das Training als ein Optimierungsverfahren vorstellen, bei dem die Gewichtungen $\mathbf{w}$ des Netzes durch die Iteration über den Trainingsdatensatz angepasst werden. Das Ziel dabei ist es, die Gleichheit zwischen den Ergebnisdaten und den prognostizierten Daten des neuronalen Netzes auf die Eingabedaten zu erreichen.

Allgemein finden neuronale Netze heutzutage in verschiedenen Themengebieten Anwendung. Ein Beispiel hierzu ist die Bildverarbeitung, welche thematisch auch in dieser Masterthesis behandelt wird. Um zu verdeutlichen, wie neuronale Netze in der Bildverarbeitung verwendet werden können, folgen einige Grundlagen zur Bildverarbeitung.

2.4.2 Bildverarbeitung

Unter dem Begriff Bildverarbeitung versteht man Algorithmen, die Informationen aus einem Bild extrahieren. Ein bekanntes Beispiel ist das autonome Fahren. Hierfür werden im Auto neben Sensoren auch Kameras verbaut, die die Straße in Fahrtrichtung optisch prüfen. Bei dieser Überprüfung sollen die Algorithmen andere Verkehrsteilnehmer*innen und die Straße inklusive aller Verkehrsschilder im Bild erkennen. Der*die Softwareentwickler*in musste in der Vergangenheit viel Zeit in die Entwicklung der Algorithmen von Vorverarbeitung[21] über Segmentierung[22] bis zur Merkmalsextraktion[23] investieren. Diese Bildverarbeitungsaufgaben lassen sich ebenfalls über den Einsatz eines neuronalen Netzes lösen. Dabei ist die Leistungsfähigkeit dieser neuronalen Netze vor allem bei den wechselnden Umgebungsbedingungen im Bild deutlich höher. Ein Teil der Bildvorverarbeitung, wie beispielsweise die einheitliche Bildgröße, wird jedoch vor der Anwendung des neuronalen Netzes auf ein Bild durch konventionelle Algorithmen umgesetzt. Für detailliertere Informationen zu den konventionellen Bildverarbeitungsalgorithmen sei auf die Quelle [10] verwiesen.

[21] Vorgehensweisen um einheitliche Bildbeschaffenheiten für die Folgealgorithmen zu schaffen, sinngemäß aus [14].
[22] Vorgehensweisen zur Trennung von Objekten im Bild, sinngemäß aus [13].
[23] Algorithmen mit denen Informationen aus dem Bild extrahiert werden, wie beispielsweise die Orientierung eines Objektes, sinngemäß aus [12].

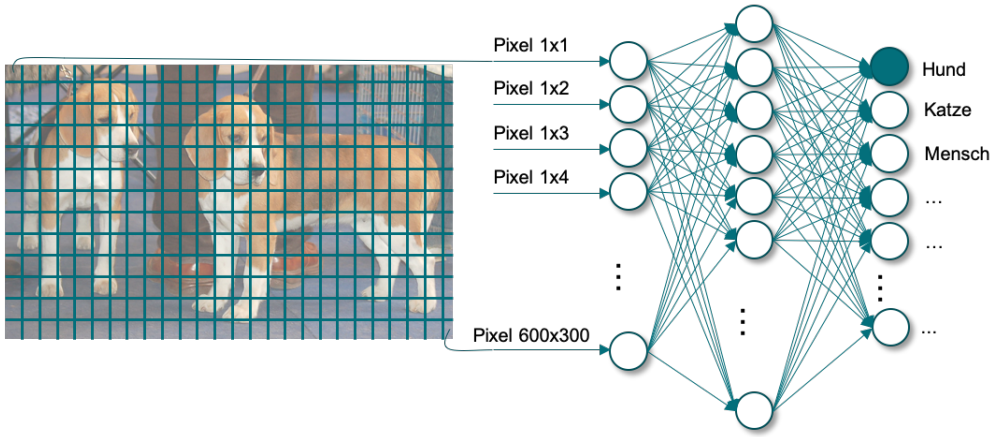

Abbildung 2.17: Eingabe eines Bildes in ein neuronales Netz mit exemplarischer Pixelstruktur, modifiziert aus [35]

In Abbildung 2.17 ist zu erkennen, wie ein Bild einen Eingabewert in ein neuronales Netz darstellen kann. Die einzelnen Pixel werden aus der Matrixform in eine Vektorschreibweise umgeformt und so in das neuronale Netz geführt. Die Anzahl der Einträge im Eingabevektor ist somit gleich der Anzahl der Pixel im Bild. In Abbildung 2.17 ist die Aufgabe der Klassifikation angedeutet.

Klassifikation

Sinngemäß nach [32] ist die Aufgabe der Klassifikation, den Inhalt eines gegebenen Bildes oder eines Bildausschnittes zu benennen. In Abbildung 2.17 ist dies bereits angedeutet. Klassifiziert wird in diesem Beispiel, dass es sich um das Objekt *Hund* auf dem Foto handelt. Um ein Netz so zu trainieren, dass dieses die Aufgabe der Klassifikation übernimmt, muss zum Einen die Anzahl der unterschiedlichen Objektklassifikationen vor dem Training im Netz eingestellt werden. Zum Anderen muss für das Training ein Datensatz erstellt werden, indem mehrere Bilder von den unterschiedlichen Objekten vorhanden sind. Zu jedem Bild muss ein Label vorliegen, mit dem der Bildinhalt betitelt wird. Der Vorgang des Benennens von mehreren Bildern nennt sich Labeling[24]. Die Größe dieser Datensätze ist von der Aufgabenkomplexität abhängig. Außerdem benötigen Netze, die bereits vortrainiert sind, deutlich weniger Trainingsdaten als ein Netz das komplett neu trainiert werden muss.

[24] englisch für Beschriftung

Objekterkennung

Die Aufgabe der Objekterkennung ist nach [19], bestimmte Objekte im Bild zu finden und die Position dieser Objekte im Bild zu bestimmen. Dabei muss der Algorithmus nicht feststellen, um welches Objekt es sich handelt, sondern lediglich wo sich diese Objekte im Bild befinden. Des Weiteren ist die Größe des Objektes wichtig, damit Folgealgorithmen, zum Beispiel die Entfernung zum Objekt, ermitteln können.

Abbildung 2.18: Objekterkennung in einem Bild, modifiziert aus [35]

In Abbildung 2.18 ist eine Objekterkennung gezeigt. Hierbei wird vom Algorithmus ein Rahmen um die erkannten Objekte gelegt. In dem Fall von Abbildung 2.18 ist außerdem eine Klassifikation direkt in die Aufgabe integriert, da die beiden umrahmten Hunde jeweils mit der Klassifikation *Hund* markiert sind. Zur Lösung der unterschiedlichen Bildverarbeitungsaufgaben existiert bereits eine große Vielzahl der Architekturen von neuronalen Netzen. So ist beispielsweise unter der Seite *paperswithcode.com* ein Ranking der State-of-the-Art[25] Algorithmen zu finden. Das Ranking bezieht sich dort auf einzelne Datensätze. Ein sehr bekannter Datensatz ist der COCO-Datensatz[26] von Microsoft, in dem Bilder von Hunden, Menschen oder auch Gegenständen, wie zum Beispiel Fahrzeugen, zu finden sind. Ein neuronales Netz, welches mit diesen Daten umgehen kann, ist sehr komplex, da in diesem Datensatz zwischen 80 verschiedenen Objektklassifikation unterschieden werden muss. Im Ranking zu diesem Datensatz[27] ist die Architektur Faster R-CNN[28] im Sommer 2016 als bestes Netz für die Aufgabe der Objekterkennung auf dem COCO-Datensatz zu finden. Im Folgenden wird auf die grobe Funktionsweise dieser Architektur eingegangen, die im Rahmen dieser Masterthesis für eine Bildverarbei-

[25] englisch für neuster Stand
[26] **C**ommon **O**bjects in **CO**ntext, englisch für gemeinsame Objekte im Kontext
[27] https://paperswithcode.com/sota/object-detection-on-coco
[28] **R**egion-based **C**onvolutional **N**eural **N**etwork, englisch für regionen- und faltungsbasiertes neuronales Netz

tungsaufgabe mit der *Tensorflow Object Detection API*, welche durch Nexperia Germany GmbH festgelegt wurde, verwendet wird.

2.4.3 Faster R-CNN

Ein Faster R-CNN dient zur Objekterkennung und Klassifikation der erkannten Objekte in einem Bild. Da in dieser Masterthesis das Faster R-CNN angewendet, jedoch nicht optimiert oder angepasst wird, ist im Folgenden nur die schematische Funktionsweise dieser Architektur erklärt. Die schlussendliche Architektur die in dieser Masterthesis verwendet wird, ist nach [35] das Context R-CNN, welches nach [5] die Architektur des Faster R-CNNs beinhaltet. Die folgenden Informationen zum Faster R-CNN sind sinngemäß aus [31] entnommen. Für weitere Einblicke in diese Architektur wird daher auf [31] verwiesen.

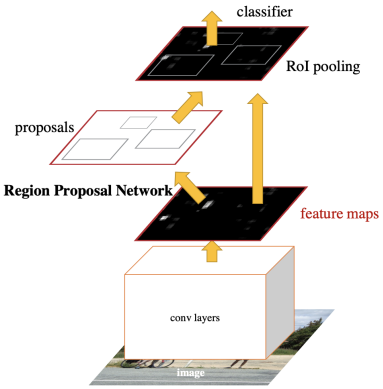

Abbildung 2.19: Schematische Funktionsweise des Faster R-CNN, übernommen aus [31]

Die Arbeitsweise dieser Architektur gliedert sich in vier verschiedene Teilaufgaben.

1. Das Filtern des Eingabebildes durch ein CNN[29].
2. Die Regionsvorhersage von Objekten durch ein RPN[30].
3. Das RoI polling[31] zur abschließenden Objekterkennung.
4. Das Durchführen der Klassifikation.

[29] **C**onvolutional **N**eural **N**etwork, englisch für faltungsbasiertes neuronales Netz
[30] **R**egion **P**roposal **N**etwork, englisch für neuronales Netz zur Regionsvorhersage
[31] **R**egion **o**f **I**nterest polling, englisch für Abfrage der interessanten Regionen

CNN

Wird ein Bild in das CNN geführt, liefert dieses als Ausgabe die sogenannte *feature map*[32]. Das CNN dient hierbei als Filter, welcher die trainierten Eigenschaften aus dem Bild hervorhebt. Ähnlich wie bei einem Kantenfilter, welcher die Kanten im Bild hervorhebt, werden durch das CNN deutlich detailtiefere Eigenschaften hervorgehoben, wie beispielsweise die Kontur eines Hundes oder eines Menschens. In Abbildung 2.19 ist die *feature map* visuell angedeutet.

RPN

Das Ziel des Algorithmus ist das Erhalten eines Rahmens um die Objekte. Nachdem das CNN auf das Bild angewendet wurde, wird die *feature map* für die Positionsvorhersage verwendet. Das RPN nutzt Masken, die auf die Aufgabe optimiert wurden, um Regionen und die ungefähre Größe des Objektes in der Region vorherzusagen. Dabei werden je nach Netzkonfiguration beispielsweise neun verschiedene Masken pro Bild verwendet um kleine, mittelgroße oder sehr große Objekte vorherzusagen. Zu jeder Objektgröße liegen in diesem Beispiel drei Rechtecke vor, welche jeweils unterschiedliche Kantenverhältnisse aufweisen.

RoI polling

Die vorhergesagten Regionen werden mit dem RoI polling an dieser Stelle abschließend analysiert. Auf jede Vorhersage aus dem RPN folgt die Anwendung eines FC[33]-Netzes. Dieses bewertet, zu wie viel Prozent es sich um ein Objekt handelt.

Abschließend werden die Objekte klassifiziert. Das Ergebnis des Algorithmus ist ein ausgegebener Vektor mit der prozentualen Wahrscheinlichkeit für die Prognose der Klassifikation. Außerdem wird der dazugehörige Objektrahmen mit den entsprechenden Bildkoordinaten ausgegeben. Diese setzen sich aus den Koordinaten der linken unteren Ecke und der Länge, sowie Breite des Rahmens zusammen.

[32] englisch für Eigenschaftskarte
[33] **F**ully **C**onnected, englisch für komplett verbunden

3 Aufgabenstellung

In diesem Kapitel wird die Aufgabenstellung für diese Masterthesis beschrieben. Dabei wird vorerst ein bestehender Messaufbau aus dem Labor von Nexperia Germany GmbH vorgestellt. Weiterführend wird der zukünftige Messaufbau definiert, welcher sich aus der Aufgabenstellung des Lastenheftes ableiten lässt. Außerdem werden die aufgeführten Aufgaben aus dem Lastenheft aufgelistet. Für einen Einblick in das verfasste Lastenheft zu dieser Masterthesis sei auf Anhang A verwiesen.

Für die Validierung von Bauteilen im Entwicklungsprozess müssen Diodenkennlinien gemessen werden. Aktuell wird diese Messung händisch von dem*der Messingenieur*in durchgeführt.

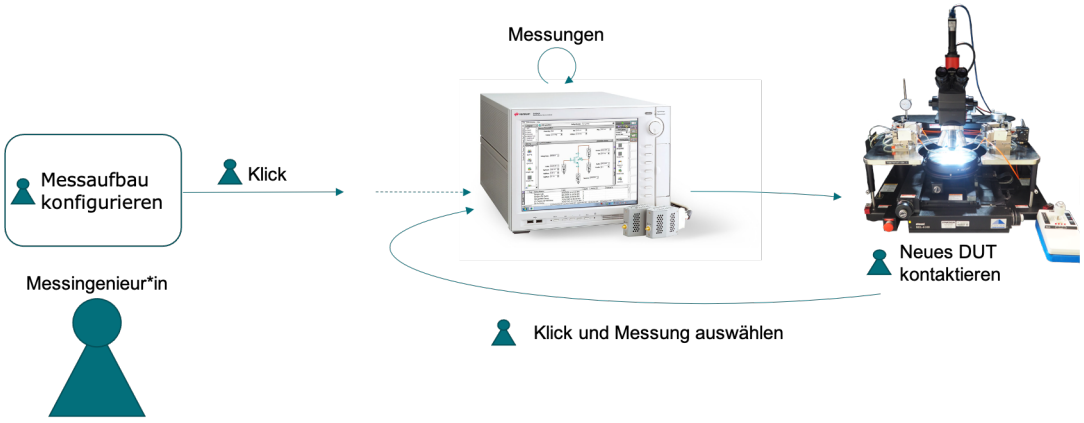

Abbildung 3.1: Existierender Messablauf für Diodenkennlinien

Der Ablauf in Abbildung 3.1 zeigt den existierenden Messablauf. Mit der Spielfigur unten links im Bild wird der*die Messingenieur*in[1], der den Messauftrag durchführt, dargestellt. Die kleineren Spielfiguren markieren die Aufgaben, die der*die Messingenieur*in während der Messung mehrerer DUT, erledigen muss. Nachdem der Aufbau konfiguriert

[1] Der*die Messingenieur*in wird in allen Abbildungen als Spielfigur dargestellt.

und das erste DUT kontaktiert ist, wird die erste Messung gestartet. Der*die Messingenieur*in ist ab diesem Schritt damit beschäftigt, neue DUT mit dem Prober zu kontaktieren und neue Messskripte auf dem SPA zu starten. In Abbildung 3.1 ist zu erkennen, dass der*die Messingenieur*in mit in die Schleife des Messablaufes eingebunden und dabei kontinuierlich beschäftigt ist.

3.1 Zielbeschreibung

Zur Entlastung des*der Messingenieurs*in und zur Optimierung des Aufbaus soll die Messung teilautomatisiert werden. Der Fokus dieser Automatisierung liegt somit auf der Kontaktierung der Messspitzen auf dem DUT. Durch die Optimierung des Darkboxprobers werden die Leerlaufzeiten am Wochenende und in der Nacht als Messzeit gewonnen, wodurch die Auslastung dieses Messplatzes steigt.

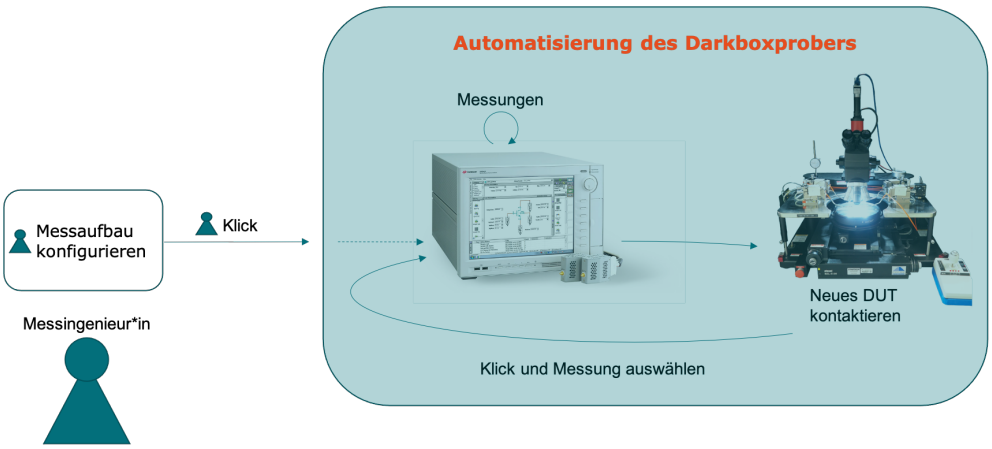

Abbildung 3.2: Ziel der Automatisierung des Darkboxprobers

Für den neuen Messaufbau obliegt die Konfiguration weiterhin dem*der Messingenieur*in. Wie in Abbildung 3.2 angedeutet, soll nach der Messung mit dem SPA automatisch das nächste Bauteil mit den Messspitzen kontaktiert werden. Sobald diese Kontaktierung abgeschlossen ist, kann dieses Bauteil gemessen werden. Dieser Ablauf soll sich für alle DUT aus dem jeweiligen Messauftrag wiederholen. Der Vergleich von Abbildung 3.2 und Abbildung 3.1 zeigt, dass der*die Messingenieur*in nicht mehr in die Messschleife integriert ist und somit entlastet wird. Die folgenden Unteraufgaben sind aus dem Lastenheft (unter Anhang A und auf der DVD zu dieser Masterthesis) wiedergegeben.

1. Es muss eine Steuerung zur automatisierten Kontaktierung von DUT mit Messspitzen realisiert werden.
2. Die optische Erkennung muss eine Genauigkeit von mindestens 70 Prozent aufweisen.
3. Es müssen sowohl DUT auf dem Wafer als auch gesägte, freiliegende DUT automatisiert kontaktiert werden können.
4. Alle Messskripte des SPAs müssen automatisiert am kontaktierten DUT durchgeführt werden können.
5. Die Software muss eine Möglichkeit zur Auswahl der zu messenden DUT bieten.
6. Die Software muss die Möglichkeit zur Auswahl der durchzuführenden Messungen auf dem SPA bieten.

Zusammenfassend ist die Aufgabe dieser Masterthesis die Entwicklung einer prototypischen Software. Diese soll eine zuverlässige Positionierung der Messspitzen an DUT auf einem Wafer oder an freiliegenden DUT realisieren. Dazu soll der Chuck zum Verfahren der Bauteile oder des Wafers gesteuert werden. Zu der automatischen Positionierung sollen zusätzlich die Messungen am DUT automatisiert werden. Dafür wird der SPA zum Durchführen der Messung gesteuert. Über eine Schnittstelle soll der*die Messingenieur*in die zu messenden DUT auswählen und den automatischen Messablauf starten können.

4 Konzept

Aus der Aufgabenstellung des Lastenheftes folgte ein ausgearbeitetes Pflichtenheft unter Anhang B. Zusätzlich wurde aus dem Pflichtenheft ein Aufgabenplan für das Projekt abgeleitet, welcher unter Anhang C zu finden ist. Mit dem Aufgabenplan entstand außerdem das Konzept für die Software, welches in diesem Kapitel vorgestellt wird. Zu den einzelnen zu entwickelnden Modulen, wird im Folgenden das Modulkonzept beschrieben.

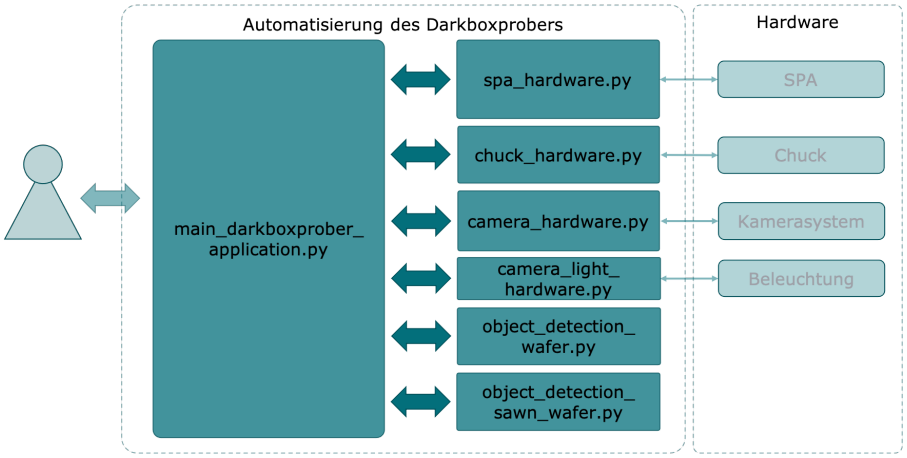

Abbildung 4.1: Konzept der Automatisierung des Darkboxprobers

Mit der GUI, die links in Abbildung 4.1 im Modul *main_ darkboxporber_ application.py* zu finden ist, wird der*die Messingenieur*in zukünftig die Anwendung bedienen. Wichtige Daten, wie die Auswahl der Messskripte, die Auswahl der zu messenden DUT und ob es sich um einen Wafer oder um vereinzelte DUT handelt, gibt der*die Messingenieur*in in diese Schnittstelle ein. Über eine Fortschrittsanzeige ist dabei der Auftragsstatus in der GUI angezeigt. Hierfür sind die abgearbeiteten DUT und Reticles aus dem Auftrag in der Wafermap farblich gekennzeichnet.

Im Folgenden werden die einzelnen Modulkonzepte erläutert, die in Abbildung 4.1 als Schnittstelle zwischen der Hardware und der GUI zu finden sind. Die GUI ist im Weiteren als zentrales Steuerelement der Software zu sehen.

4.1 SPA

Mit dem Modul *spa_ hardware.py* wird das Handbuch des SPAs zur VISA-Schnittstelle programmiert. Dieses Modul wird Funktionen zum Auslesen existierender Messskripte, zum Auswählen bestimmter Messskripte und zum Benennen der durchgeführten Messung auf dem SPA bieten. Über diese Beispiele hinaus, werden alle weiteren Kommandos aus dem Handbuch programmiert, um für zukünftige Erweiterungen verfügbar zu sein. Ein weiterer Inhalt dieses Moduls wird eine Funktion zum Überprüfen des Messgerätestatus sein. Zusammenfassend ist dieses Modul für den Zugriff auf den SPA zuständig. Zusätzlich beinhaltet dieses in der Anwendung alle notwendigen Funktionen zur Bedienung des SPAs.

4.2 Chuck

Mit dem Modul *chuck_ hardware.py* wird das Handbuch des Chucks zur VISA-Schnittstelle programmiert. Eine Methode in diesem Modul gibt als Rückgabewert die Position des Chucks zurück. Des Weiteren realisieren einige Methoden den Positionswechsel des Chucks über eine relative Fahrt oder über vorgegebene Koordinaten für die X- und Y-Achse. Auch in diesem Modul wird der Status des Chucks abgefragt, um diesen nicht mit VISA-Kommandos zu überlasten. Die Steuerung der Z-Achse ist ebenfalls ein Bestandteil dieses Moduls, um jederzeit die Kontaktierung des DUT durchführen zu können. Damit stets die gleichen Anfangsbedingungen herrschen, wird eine Initialisierungsroutine mit Systemkommandos Bestandteil dieses Moduls sein.

4.3 Kamera

Das Modul *camera_ hardware.py* wird die Schnittstelle zur eingebauten Kamera realisieren. Mit einer implementierten Methode wird das aktuelle Bild aus der Kamera gelesen. Dieses Bild dient gleichzeitig als Rückgabewert der Methode. Außerdem beinhaltet das

Modul einen Stream, bei dem über OpenCV eine direkte Ausgabe des Live-Bildes erfolgt. Die Stream-Funktion wird für die Debuggingphase des Chucks benötigt, um die Fahrbefehle optisch zu kontrollieren. Damit die Software auch unabhängig von gegebener Hardware entwickelt und getestet werden kann, beinhaltet das Modul ein Exception-Handling. Sollte die Kamera nicht am System angeschlossen sein, wird ein Ersatzbild zum aktuellen Bildframe als Funktionsrückgabe verwendet.

4.4 Licht

Mit dem Modul *camera_ light_ hardware.py* werden zwei Funktionen realisiert. Diese dienen zum An- und Ausschalten der Beleuchtung.

4.5 Benutzerschnittstelle

Die Benutzerschnittstelle wird in mehreren einzelnen Modulen entwickelt. Als Hauptmodul ist *main_ darkboxprober_ application.py* vorgesehen, welches die einzelnen Module innerhalb der GUI initialisiert und verknüpft.

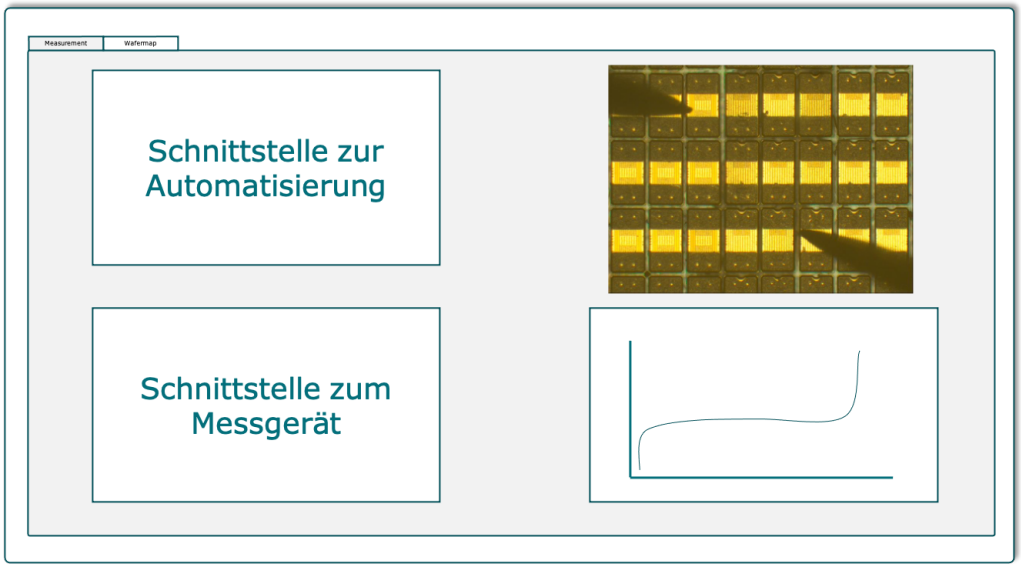

Abbildung 4.2: Konzept der GUI Teil 1

In Abbildung 4.2 ist das Konzept zur ersten Seite der GUI zu erkennen. Umgesetzt in *automation_ screen_ ui.py* ist dabei das Teilmodul zur Realisierung der Schnittstelle zur Automatisierung. In diesem Bereich kann der*die Nutzer*in die automatisierte Messung starten und relevante Einstellungen tätigen. Unten links wird die Schnittstelle zum Messgerät implementiert, über die der*die Nutzer*in das Messgerät einstellt und die durchzuführenden Messskripte auswählt. Dieses Teilmodul wird in *measurement_ screen_ ui.py* realisiert. Oben rechts ist das Bild der Kamera im Teilmodul *camera_ screen_ ui.py* abgebildet, sodass der*die Nutzer*in im Betrieb prüfen kann, ob die Kontaktierung der Messspitzen auf den Pads erfolgreich durchgeführt wurde. Abschließend werden unten rechts alle erhaltenen Messergebnisse dargestellt. Dieses Teilmodul wird in *plot_ screen_ ui.py* umgesetzt.

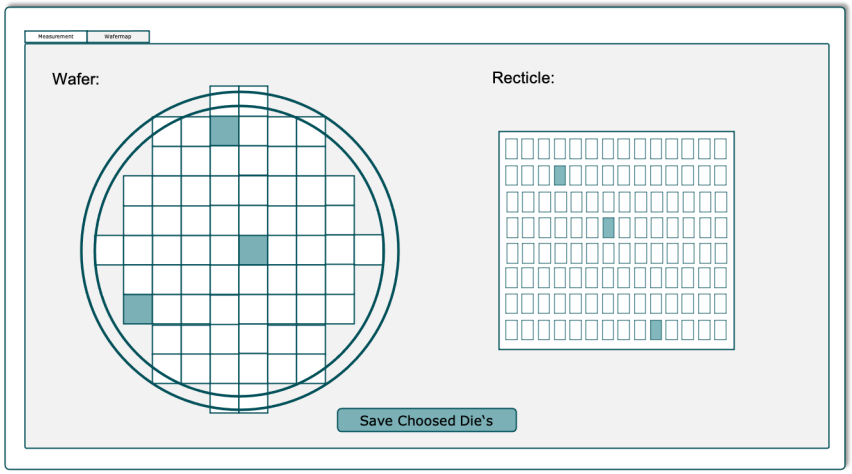

Abbildung 4.3: Konzept der GUI Teil 2

Auf der zweiten Seite der GUI ist eine Schnittstelle zur Auswahl der zu messenden DUT geboten. Es werden, wie links zu erkennen ist, die zu messenden Reticles ausgewählt. Zusätzlich sind die speziellen DUT rechts ausgewählt, zu denen der Messauftrag vorliegt. Dabei wiederholt sich die Messung jedes ausgewählten DUT in jedem ausgewählten Reticle. Die Umsetzung dieses Teilmoduls erfolgt in *wafermap_ screen_ ui.py*. Weitere Teilmodule, die für die Umsetzung dieser Grundfunktionalitäten der GUI notwendig sind, werden in der Entwicklungsphase zum System hinzugefügt.

4.6 Bildverarbeitung

Für die Positionierung der Messspitzen an einem DUT auf dem Wafer ist das sogenannte Stepping eine gängige und präzise Variante. Hierbei wird der bekannte Pitch des Wafers dazu verwendet, die Positionen der DUT auf dem Wafer zu ermitteln und nacheinander zu kontaktieren. Die Voraussetzung zur exakten Kontaktierung mit dem Stepping ist zum Einen die Arretierung des Wafers über die Notch. Zum Anderen darf sich der Wafer durch eine Temperaturveränderung nicht verformen, da sich sonst der Pitch über den Wafer verändern könnte. An der Darkbox sind die Armaturen zur Ausrichtung des Wafers nicht vorhanden, sodass diese händisch durchgeführt werden muss. Die Genauigkeit der händischen Arretierung schwankt je nach Nutzer*in, sodass beim Stepping über den Wafer die Kontaktierung am DUT nicht gewährleistet werden kann. Neben dem Wafer werden in dieser Anwendung auch freiliegende DUT kontaktiert. Für die Umsetzung dieser Aufgaben wird daher ein Bildverarbeitungsalgorithmus zur optischen Erkennung der DUT auf dem Wafer oder freiliegender DUT entwickelt. Genauer wird als Bildverarbeitungsalgorithmus ein neuronales Netz zur Objekterkennung mit der Tensorflow Object Detection API verwendet. Diese API bietet die Möglichkeit zur Einbundung und Verwendung von verschiedenen Netzarchitekturen. Die Grundlage der API ist das bereits unter Unterabschnitt 2.4.3 beschriebene Verfahren des Faster R-CNNs. Über die Module *object_detection_wafer.py* und *object_detection_sawn_wafer.py* wird die API in die Anwendung integriert.

4.6.1 Training

Für die Anwendung der Objekterkennung über die Tensorflow Object Detection API, muss diese einem Training unterzogen werden.

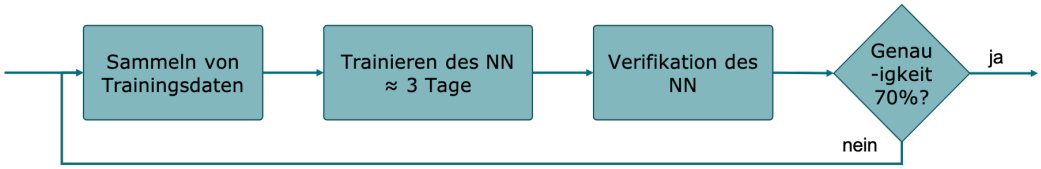

Abbildung 4.4: Konzept des Trainings des neuronalen Netzes

In Abbildung 4.4 ist der Trainingszyklus zu erkennen. Es werden Trainingsdaten gesammelt, ein circa dreitägiges Training des Netzes durchgeführt und abschließend zu

diesem Zyklus eine Verifikation durchgeführt. Sobald diese eine Genauigkeit von 70 Prozent oder höher aufweist, ist die Aufgabe des Trainings erfüllt. Sollte der Zyklus eine geringere Genauigkeit als 70 Prozent aufweisen, dient als Strategie vorerst die Vergrößerung des Trainingsdatensatzes. Es wird sichergestellt, dass keine doppelten Bilder im Trainingsdatensatz verwendet werden, damit kein Übertraining auf spezielle Bildeigenschaften stattfindet. Außerdem soll der Datensatz Bilder von unterschiedlichen Wafern beinhalten. Das Training des Netzes wird nach der Erweiterung des Trainingsdatensatzes erneut durchgeführt. Die Trainingsdatenerweiterung dient als Strategie für sieben Trainingszyklen. Sollte das Ziel in diesen sieben Zyklen nicht erreicht werden, dient die Verwendung von weiteren Netzarchitekturen in der API als Ausweichstrategie. Für die neue Netzarchitektur wird der Zyklus zur Datenerweiterung wiederholt.

Die gesamte *Tensorflow Object Detection API* wird in die Anwendung integriert. Dabei gilt die Unterscheidung zwischen den zwei Anwendungsbereichen Wafer und freiliegende DUT. In dieser Masterthesis wird in Kapitel 6 das Trainings als Ganzes betrachtet, wobei nicht detailliert auf die einzelnen Trainingsintervalle eingegangen wird.

4.6.2 Wafer

Wie bereits erwähnt, kontaktiert man die DUT auf einem Wafer normalerweise über das Stepping nacheinander. Wenn jedoch die Arretierung des Wafers nicht ausreichend genau ist, kann eine Fehlkontaktierung die Folge sein.

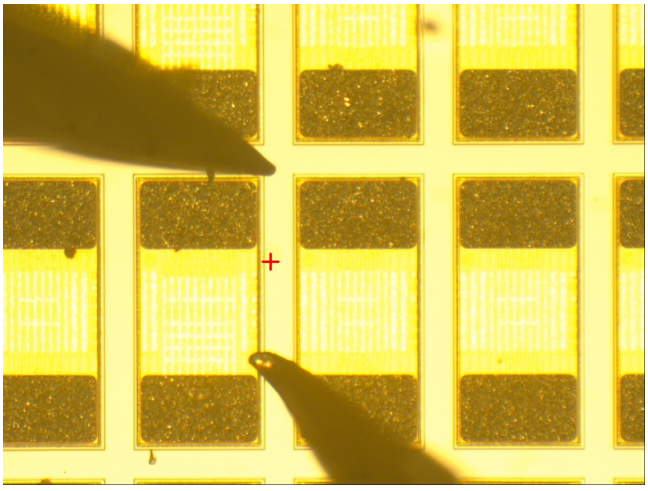

Abbildung 4.5: Fehlkontaktierung der Messspitzen auf dem Wafer

In Abbildung 4.5 ist eine exemplarische Fehlkontaktierung abgebildet. Neben der falschen Arretierung des Wafers, kann dies ebenfalls durch falsch eingegebene Werte vorkommen, wie beispielsweise der Pitch des Reticles oder der Pitch des DUT. Das Ziel der Objekterkennung beim Stepping über den Wafer ist es, diese Fehlkontaktierungen zu korrigieren.

Für die Korrektur sind zwei Eingaben notwendig. Zum Einen wird der Mittelpunkt x_{ref} und y_{ref} des kontaktierten DUT benötigt. Diesen markiert der*die Nutzer*in nach der ersten händischen Kontaktierung mit dem roten Kreuz im Bild. Zum Anderen benötigt man die Skalierung x_{scale} und y_{scale}. Dieser Wert verknüpft das Kamera- mit dem Chuckkoordinatensystem. Die Skalierung ist aus zwei Eingaben von dem*der Nutzer*in zu ermitteln. Als Erstes gibt der*die Nutzer*in den Pitch des DUT $x_{pitch,real}$ und $y_{pitch,real}$ ein. Außerdem rahmt dieser*diese ein DUT inklusive einer Sägebahn im Bild ein, wobei der Pitch des DUT $x_{pitch,cam}$ und $y_{pitch,cam}$ in Kamerakoordinaten bestimmt wird. Die Skalierung ergibt sich aus den folgenden zwei Formeln.

$$x_{scale} = \frac{x_{pitch,real}}{x_{pitch,cam}} \tag{4.1}$$

$$y_{scale} = \frac{y_{pitch,real}}{y_{pitch,cam}} \tag{4.2}$$

Für jede Kontaktierung nach der Erstkontaktierung wird die Objekterkennung auf das aktuelle Bild angewendet. Zur Korrektur dient die ermittelte Mitte des DUT x_{actual} und y_{actual} aus dem aktuellen Bild. Die Korrektur ergibt sich aus den folgenden Gleichungen.

$$x_{correction} = (x_{actual} - x_{ref}) \cdot x_{scale} \tag{4.3}$$

$$y_{correction} = (y_{actual} - y_{ref}) \cdot y_{scale} \tag{4.4}$$

Mit $x_{correction}$ und $y_{correction}$ ergeben sich die Werte, welche für eine relative Fahrt mit dem Chuck die Korrekturfahrt realisieren.

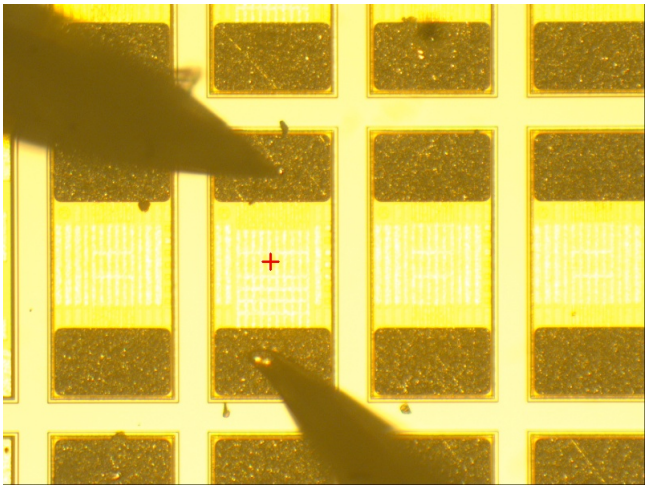

Abbildung 4.6: Kontaktierung nach der Korrektur auf dem Wafer

In Abbildung 4.6 ist die Kontaktierung nach der Korrektur abgebildet. Das Kreuz befindet sich jetzt in der Mitte des DUT. Nachdem die relative Bewegung des Chucks durchgeführt ist, wird der Wafer zur Kontaktierung in die Kontaktierungsebene gefahren.

4.6.3 Freiliegende DUT

Zur Ermittlung der Position des nächsten freiliegenden DUT im Koordinatensystem des Chucks dient die optische Erkennung. Hierbei gelten die gleichen Rechengesetze wie schon in Unterabschnitt 4.6.2 erwähnt.

Auch bei den freiliegenden DUT wird das aktuelle Bild durch das neuronale Netz ausgewertet. Dabei entsprechen jedoch x_{actual} und y_{actual} dem nächsten zu kontaktierenden DUT.

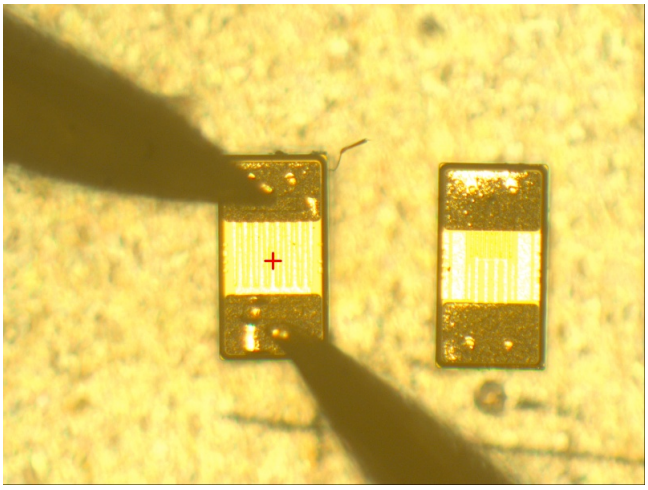

Abbildung 4.7: Kontaktierung der freiliegenden DUT

In Abbildung 4.7 ist das von dem*der Nutzer*in gesetzte rote Kreuz zu erkennen. Dieses markiert wieder die Werte x_{ref} und y_{ref}. Die Anforderung an die Objekterkennung ist, das nächste DUT rechts von dem Kontaktierungspunkt zu erkennen und diese Koordinaten zur Verfügung zu stellen. Über die Umrechnung können die realen Koordinaten bestimmt und über eine relative Fahrt mit dem Chuck unter den Messnadeln positioniert werden.

Für diese Funktionalität ist es wichtig, dass der*die Messingenieur*in das nächste zu kontaktierende DUT so im Messaufbau positioniert, dass dieses im Kamerabild zu sehen ist, während das vorherige DUT kontaktiert ist. Außerdem darf die Ausrichtung des DUT nicht willkürlich sein. Der*die Messingenieur*in muss die DUT so nebeneinander positionieren, dass die Messspitzen über eine X-/Y-Fahrt auf die Pads passen. Der*die Messingenieur*in muss die DUT in einem Winkel von circa $\pm 15°$ zueinander positionieren.

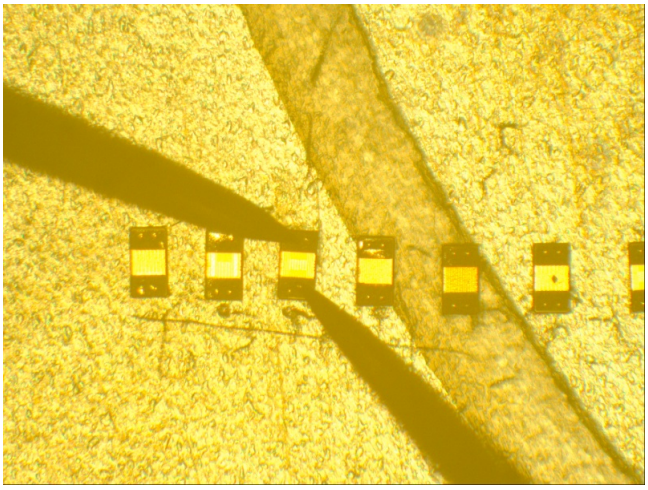

Abbildung 4.8: Aufgereihte DUT für die automatisierte Messung

Wie in Abbildung 4.8 exemplarisch gezeigt wird, muss der*die Messingenieur*in die DUT in einer Reihe anordnen, sodass der Chuck die Messreihe nach rechts durcharbeiten kann. Die DUT müssen nach Augenmaß gerade in einer Reihe angeordnet sein, sodass die Messspitzen über eine X-/Y-Korrekturfahrt weiterhin auf die Pads passen. Eine Korrektur über die rotative Achse des Chucks ist nicht vorgesehen, da dieser lediglich einen Wirkungsbereich von $-8°$ bis $+8°$ bereitstellt.

In diesem Kapitel wurde das Konzept zur Umsetzung dieser Masterthesis vorgestellt. Die automatische Kontaktierung von DUT ist dabei das grundlegende Ziel der Automatisierung. Dabei ist zum Einen die Kontaktierung von DUT auf dem Wafer und zum Anderen von vereinzelten DUT automatisiert. Es folgt die Umsetzung dieses Konzeptes mit der Beschreibung im folgenden Kapitel.

5 Implementierung

Aus dem Konzeptplan wurde für die Umsetzung ein Aufgabenplan formuliert, der in Anhang C zu finden ist. Die entwickelten Module zur Realisierung der verschiedenen Aufgaben aus dem Aufgabenplan werden im Folgenden beschrieben. Hierfür spiegeln Klassendiagramme und Ablaufdiagramme den Inhalt der einzelnen Module wieder. Beginnend werden die Inhalte zu den Modulen der Hardwarekomponenten in Abschnitt 5.1 bis Abschnitt 5.4 vorgestellt. Die Ablaufdiagramme zu diesen Modulen sind dabei jeweils aus der Modulsicht zu sehen, wobei das jeweilige Python-Programm importiert und verwendet wird. Außerdem werden wichtige Programmzeilen in Listings vorgestellt, wobei diese zur Lesbarkeit abgekürzt werden. Für den originalen Quelltext zu dieser Masterthesis sei auf die DVD in der gedruckten Ausgabe dieser Masterthesis verwiesen. Weiterführend wird in diesem Kapitel die entwickelte Benutzerschnittstelle auf der Ebene des*der Nutzer*in beschrieben. Die Abbildung aller Klassendiagramme visualisiert abschließend die Systemstruktur der Gesamtanwendung.

5.1 Ansteuerung des SPAs

Im Modul *spa_ hardware.py* ist der Quelltext zur Realisierung der Messgerätesteuerung zu finden. Dabei ist *pyvisa* die Bibliothek die Python zur VISA-Kommunikation zur Verfügung stellt.

5.1.1 VISA-Kommunikation

Installiert wird diese Bibliothek mit dem Befehl *pip install pyvisa*. Im Folgenden wird eine exemplarische Kommunikation über die pyvisa vorgestellt.

Quelltext 5.1: Implementation des Python-Paketes pyvisa, modifiziert aus [18]

```
1 import pyvisa
2 DVC_NM = "TCPIP0:192.168.0.210::5025::SOCKET"
3 if __name__ == "__main__":
4     rm = pyvisa.ResourceManager()
5     b1500a = rm.open_resource(DVC_NM, read_termination="\n")
6     device_id_str = b1500a.query("*IDN?")
7     print(device_id_str)
8 >>Agilent Technologies,Keysight EasyEXPERT,
9 >>B1500AM55231258,6.2.1927.7790
```

Nach dem Einbinden von pyvisa in Quelltext 5.1 und dem geöffneten Ressource Manager, wird die Ressource des Messgerätes geöffnet. Durch *DVC_NM* ist definiert, dass Ethernet als physikalische Schnittstelle verwendet wird. Mit read_termination wird das Ende des zu lesenden Strings deklariert. Das Kommando *b1500a.query()* sendet den übergebenen String über VISA. Die darauffolgende Geräteantwort wird direkt durch die Methode ausgelesen und als Rückgabewert in *device_id_str* geschrieben. In Quelltext 5.1 wird das SCPI **IDN?* zum Messgerät gesendet. Der SPA antwortet auf dieses Kommando mit dem Geräteidentifikationsstring, welcher in diesem Beispiel in Zeile neun über *print()* ausgegeben wird. Häufig sind in diesem String die Seriennummer und der Gerätetyp zu finden, sodass die Kommunikation mit dem richtigen Messgerät verifiziert werden kann. Die Antwort des SPAs ist in Zeile 11 und 12 zu finden, wodurch das Messgerät über die Ressource verifiziert werden konnte.

Neben den üblichen Kommandos werden vom Hersteller für jedes VISA-Gerät spezielle Kommandos zur Gerätesteuerung entwickelt. In Quelltext 5.2 ist eines der verschiedenen gerätespezifischen Kommandos in einer Funktion implementiert.

Quelltext 5.2: Funktion zum Öffnen des Workspaces von EasyEXPERT über VISA

```
1 def open_workspace(name_of_ws):
2     spa.write(":WORKspace:OPEN {}". format(name_of_ws))
```

In Quelltext 5.2 ist zu erkennen, dass die zwei Programmzeilen eine Umsetzung des Strings in eine Funktionsdeklaration realisieren. Dieses Kommando dient zum Öffnen eines Workspaces von EasyEXPERT mit dem Namen aus dem Übergabeparameter *name_of_ws*.

Neben diesem exemplarischen Kommando existieren weitere gerätespezifische Kommandos, die in der Tabelle 5.1 aufgeführt werden. Aufgelistet sind dort jedoch nur die Kommandos, die für diese Masterthesis relevant sind.

Tabelle 5.1: Relevante VISA-Kommandos für den SPA, sinngemäß aus [34]

Kommando	Funktion
:WORKspace:CATalog?	Anfordern des Workspacekataloges
:WORKspace:CLOSe	Workspace schließen
:WORKspace:OPEN "name"	Workspace "name"öffnen
:WORKspace:STATe?	Workspace Status abfragen
:APPlication:CATalog?	Anfordern des Applicationkatalog
:APPlication:SELect "name"	Application "nameäuswählen
:PRESet:CATalog?	Anfordern des Presetgroupkataloges
:PRESet:SETup:SELect "name"	Presetgroup "nameäuswählen
:RUN:SINGle	Geöffnetes Messskript starten
:STRing "name","value"	Setzen oder Rücksetzen des Namens der Messung
:RESult:FETch?	Messdaten nach der Messung anfordern
...	...

Mit diesen VISA-Kommandos wird die Software EasyEXPERT bedient, welche die Messung auf dem SPA steuert. Zum Beispiel startet der*die Nutzer*in EasyEXPERT mit dem Auswählen des Workspaces, in dem die einzelnen Messskripte auf dem Messgerät zu finden sind. In Analogie dazu werden mit *:WORKspace:CATalog?* die existierenden Workspaces abgefragt und können, wie bereits in Quelltext 5.2 vorgestellt, über das Kommando *:WORKspace:OPEN "name"* geöffnet werden. Der Platzhalter *name* muss dabei mit einem existierenden Namen, wie zum Beispiel *Initial Workspace*, ersetzt werden. Alle hier aufgelisteten Kommandos werden, wie in Quelltext 5.2 exemplarisch dargestellt, implementiert. Für einen modularisierten Quelltext werden die einzelnen Funktionen, wie in Quelltext 5.1 und Quelltext 5.2, jedoch in Methoden umgesetzt und in der Klasse *KeysightB1500A* realisiert.

5.1.2 Klasse *KeysightB1500A*

In *spa_hardware.py* ist die Klasse *KeysightB1500A* zu finden, welche in dem folgenden UML[1]-Klassendiagramm dargestellt ist.

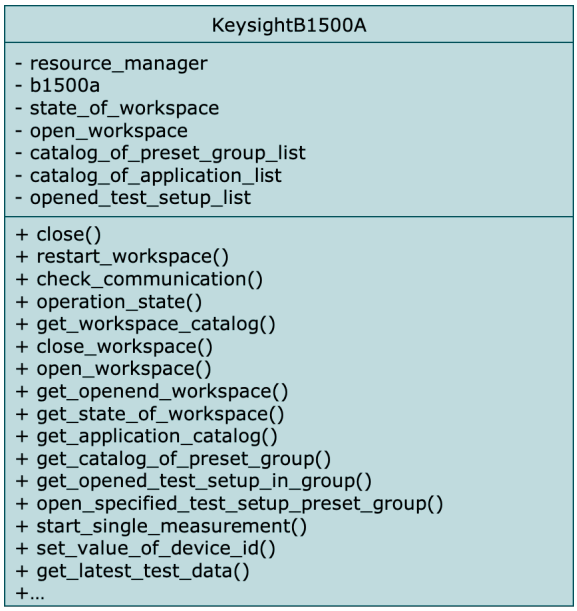

Abbildung 5.1: Klassendiagramm zum Modul *spa_hardware.py*

In dem Klassendiagramm in Abbildung 5.1 sind im oberen Bereich die Attribute erkennbar. Beispielsweise existiert dort die Variable *open_workspace*, welche den Namen des geöffneten Workspaces beinhaltet. In *b1500a* wird die Ressource der VISA-Schnittstelle abgelegt. Außerdem sind die verschiedenen Methoden für die Steuerung von EasyEXPERT im unteren Bereich des Klassendiagramms zu finden. Hinter jeder Methode steht wie in Quelltext 5.2 ein String, der über den *query*- oder *write*-Befehl zum SPA gesendet wird.

Mit zwei verschiedenen Abläufen wird dieses Modul in das Gesamtprogramm eingebunden. Der erste Ablauf beinhaltet die Einstellung des SPAs, der zweite Ablauf die Durchführung der Messung.

[1] **U**nified **M**odeling **L**anguage, englisch für vereinheitlichte Modellierungssprache

5.1.3 Ablauf 1 - Einstellen des SPAs

In der Konfigurationsphase der Automatisierung wird eine Sequenz zur Auswahl der Messungen durchlaufen. Hierbei wird der*die Nutzer*in die speziellen Messskripte anwählen, die auf dem DUT für seinen spezifischen Messauftrag ausgeführt werden sollen. Die Sequenz ist in Abbildung 5.2 als Ablaufdiagramm festgehalten.

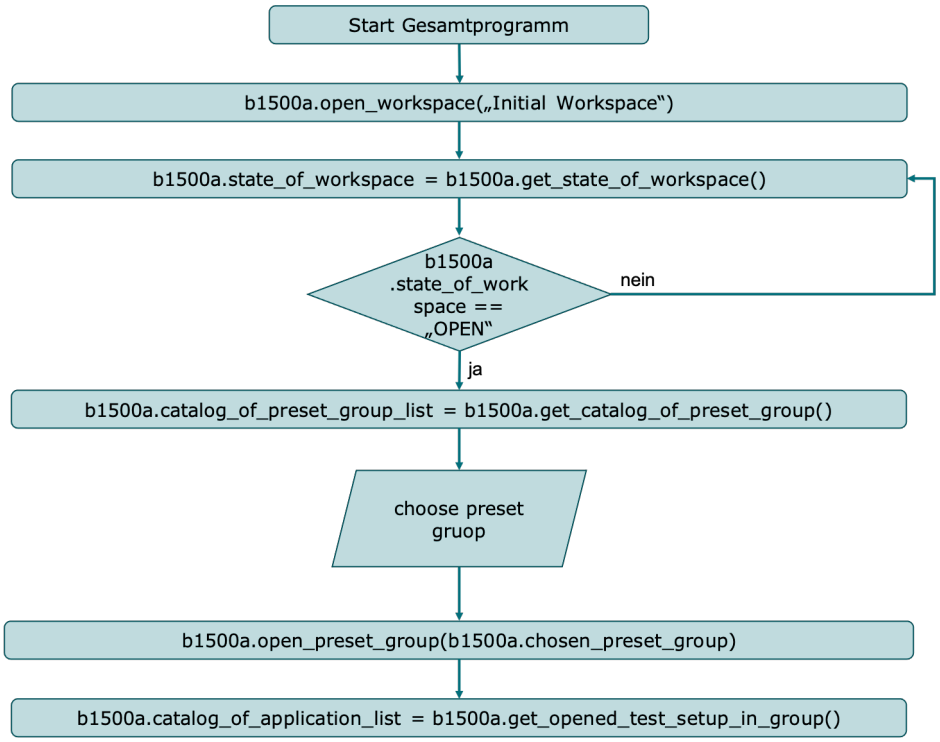

Abbildung 5.2: Ablaufdiagramm der Startkommunikation mit dem SPA

Beim Start des Gesamtprogramms wird der *Initial Workspace* von EasyEXPERT geöffnet. Dieser Ablauf dauert einige Sekunden bis Minuten. Dabei wird mit der Methode *open_workspace* der Befehl zum Öffnen des Workspaces gesendet. Über *get_state_of_workspace* wird abgefragt, ob das Öffnen durchgeführt wurde. Sobald die Rückmeldung *OPEN* über VISA gelesen wird, kann das nächste Kommando gesendet werden. Mit *get_catalog_of_preset_group* wird der Katalog der Preset Group aus dem SPA angefordert. Dieser wird in der Variable *catalog_of_preset_group_list* abgelegt, wobei der Katalog als eine Liste mit Strings vorliegt. Jeder String in dieser Liste beinhaltet einen Namen einer Preset Group aus EasyEXPERT. Daraufhin muss aus dem Katalog eine Preset Group ausgewählt werden. Zu dieser Gruppe wird anschließend der vorhandene Katalog

der Messskripte über *get_ opened_ test_ setup_ in_ group* angefordert, welcher wiederum in *catalog_ of_ application_ list* abgelegt wird. Auch diese Variable beinhaltet eine Liste mit Strings, welche die Namen der einzelnen Messskripte in der ausgewählten Preset Group umfasst.

Standardmäßig wird beim Softwarestart der Katalog der Messskripte aus der ersten Preset Group in *catalog_ of_ preset_ group_ list* angefordert. Anschließend kann der*die Nutzer*in über ein erneutes Aufrufen der Methode *get_ catalog_ of_ preset_ group* die Anfrage der existierenden Messskripte wiederholen, sodass diese im laufenden Betrieb stets aktualisiert werden können. Der*die Nutzer*in kann die Methode über eine Funktion in der GUI aufrufen.

Zusammenfassend werden mit der Startsequenz alle wichtigen Funktionen zur Messskriptauswahl aus dem SPA verbunden. Die Auswahl der Skripte wird letztendlich über die GUI durchgeführt, wobei dem*der Nutzer*in *catalog_ of_ application_ list* angezeigt wird. Das Prinzip ist somit direkt aus dem SPA übernommen, wodurch dem*der Messingenieur*in eine gewohnte Bedienung geboten wird.

5.1.4 Ablauf 2 - Durchführen der Messungen mit dem SPA

Der zweite Ablauf beginnt sobald die Konfiguration der Messung abgeschlossen ist und der*die Nutzer*in die automatisierte Messung bereits gestartet hat. Die Auswahl der durchzuführenden Messskripte und das Vorliegen der Liste *measurement_ list* stellen den Ausgangspunkt dar. Die Sequenz wird als Ablaufdiagramm in Abbildung 5.3 dargestellt.

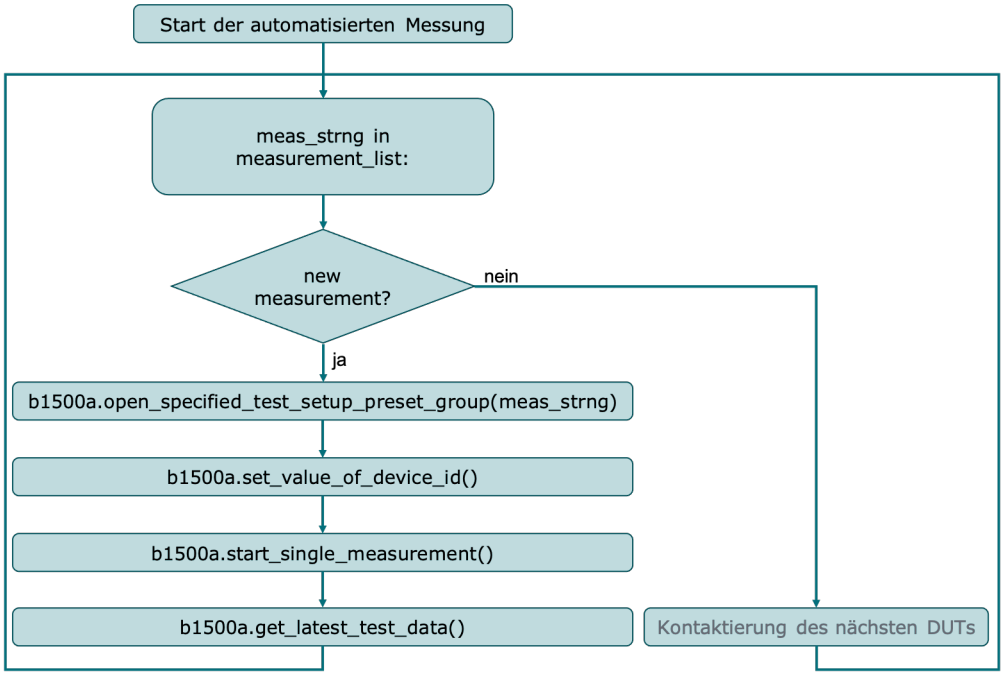

Abbildung 5.3: Durchführung der einzelnen Messungen am DUT

Nach dem Start der automatisierten Messung ist das erste DUT bereits kontaktiert, sodass zunächst die gesamten Messungen am DUT durchgeführt werden müssen. Hierfür wird die Liste *measurement_list* mit einer for-Schleife durchlaufen. Es wird das jeweilige Messskript in *meas_strng* über die Methode *open_specified_test_setup_preset_group* geöffnet. Über *set_value_of_device_id* wird das nächste Messergebnis im Voraus benannt. Die Methode *start_single_measurement()* wird das ausgewählte Messskript an dem DUT starten. Für die Messdatenablage gibt es zwei Varianten. Zum Einen legt der SPA die Daten auf einem internen Verzeichnis ab, welches nach Beendigung des Messauftrages geleert werden kann. Zum Anderen können mit der Methode *get_latest_test_data()* die entstandenen Messdaten im Automatisierungsablauf ausgelesen werden. Sobald die Liste *measurement_list* vollständig abgearbeitet wurde, kann das nächste DUT kontaktiert werden. Dieser Prozess wird für die Gesamtheit der zu messenden DUT wiederholt. Die Kontaktierung des nächsten DUT wird nicht in dem SPA-Modul durchgeführt, weshalb diese Aktion in Abbildung 5.3 in grau dargestellt ist.

5.2 Steuerung des Chucks

In dem Modul *chuck_hardware.py* wird die Steuerung des Chucks realisiert, wobei die Bibliothek *pyvisa* erneut die Grundlage der Kommunikation bildet. Dabei werden wichtige Kommandos aus der Kommunikation aufgelistet, die Initialisierungsroutine vorgestellt und die Einbindung des Moduls in den Ablauf einer Messung beschrieben.

5.2.1 VISA-Kommunikation

Ähnlich wie in Quelltext 5.1, wird auch die VISA-Schnittstelle des Chucks initialisiert. Hierbei wird Zeile drei aus Quelltext 5.1 mit Quelltext 5.3 ersetzt, da es sich beim Chuck um einen neuen VISA-Teilnehmer mit einer anderen Adresse handelt.

Quelltext 5.3: VISA-Adresse des Chucks

```
1 DVC_NM = "GPIB1::5::INSTR"
```

Quelltext 5.3 beinhaltet somit den Adressstring in dem GPIB als physikalische Schnittstelle für diesen VISA-Teilnehmer festgelegt ist. Auch diese Schnittstelle muss verifiziert werden, wobei das VISA-Kommando **IDN?* verwendet wird.

Quelltext 5.4: Antwort des Chucks auf das VISA-Kommando **IDN?*

```
1 >>C5 "UNSUPPORTED Command" *IDN?
```

Die Antwort des Chucks auf das **IDN?*-Kommando ist in Quelltext 5.4 zu finden, wodurch eine Verifikation des Gerätes über dieses Kommando nicht möglich ist. Ersatzweise werden Kommandos zum Bewegen des Chucks über VISA gesendet, damit die Schnittstelle verifiziert werden kann. Beispiele hierzu findet man in Tabelle 5.2, welche sinngemäß aus der Gerätedokumentation [28] entnommen wurden.

Tabelle 5.2: Relevante VISA-Kommandos für den Chuck, sinngemäß aus [28]

Kommando	Funktion
	Bewegungsbefehle
MP D X %	Bewegung der X-Achse zu gegebener Position
MP D X % Y %	Bewegung der X- und Y-Achse zu gegebener Position
MP D T %	Bewegung der T-Achse zu gegebener Position
MH D	Bewegung zur Home Position
MR D X % Y %	Relative Bewegung der X- und Y-Achse
MR D T %	Relative Bewegung der T-Achse
MZ D C	Bewegung der Z-Achse in die Kontaktierungsebene
MZ D S	Bewegung der Z-Achse in die separate Ebene
MK D X	Bewegung der X-Achse abbrechen
	Abfragen
QP D	Positionsabfrage
	Einstellungsbefehle
SZ D AUTO N	Automatische Bewegung der Z-Achse deaktivieren
SZ D AUTO E C	Automatische Bewegung der Z-Achse aktivieren
SS M F	Kommandos zurücksetzen
...	...

Die Prozentzeichen in Tabelle 5.2 sind Platzhalter für einen Wert, der im jeweiligen Kommando übergeben wird. Ähnlich wie beim SPA, werden diese Kommandos in Methoden der Klasse *ChuckControl* implementiert, sodass der Chuck objektorientiert in das Gesamtprogramm eingebunden werden kann.

Quelltext 5.5: Umsetzung der Kommandos zur Chuckkommunikation in Methoden

```
1 def move_relative_xy(self, x_val, y_val):
2     self.chuck.write("MR D X {} Y {}". format(x_val, y_val))
```

In Quelltext 5.5 ist exemplarisch dargestellt, wie ein Kommando in *ChuckControl* zu einer Methode umgesetzt wird. Zu jedem der in Tabelle 5.2 gelisteten Kommandos, findet sich eine Methode in der Klasse *ChuckControl*. Außerdem sind weitere Methoden, die für diese Masterthesis nicht relevant sind, aus [28] in *ChuckControl* zu finden.

5.2.2 Klasse ChuckControl

Ähnlich wie beim SPA, ist auch in diesem Modul der *resource_manager* als Attribut zu finden. Das Attribut *chuck* ist dabei die Ressource der VISA-Schnittstelle. Auf diese Ressource können wieder die bekannten Befehle *write()*, *query()* und viele weitere Kommandos aus pyvisa angewendet werden.

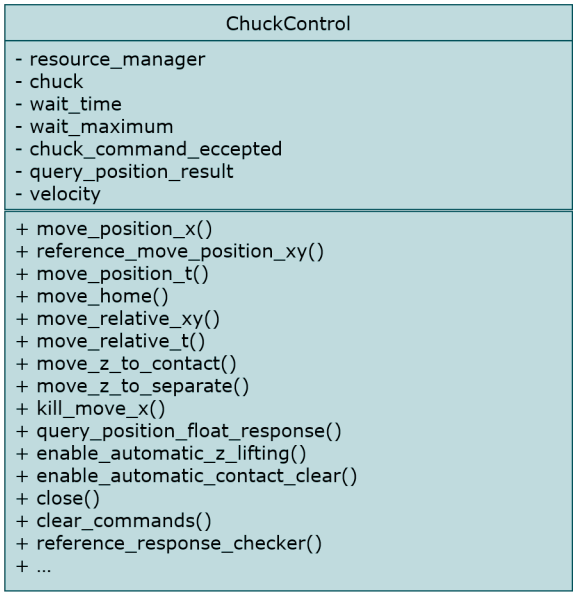

Abbildung 5.4: Klassendiagramm der Klasse *ChuckControl* zum Modul *chuck_hardware.py*

Mit dem Attribut *wait_time()* aus Abbildung 5.4 wird eine Wartezeit nach jedem VISA-Kommando bestimmt. In *wait_maximum()* wird eine maximale Wartezeit für ein VISA-Kommando festgelegt. Mit dieser Variable werden verschiedene Fehlerquellen abgefangen, die während der Kommunikation mit dem Chuck vorkommen können. Außerdem ist die Methode *reference_move_position_xy* zum Abfangen der Fehlerquellen zuständig. Diese Methode wird im Ablaufdiagramm in Abbildung 5.5 vorgestellt. Des Weiteren wird dort gezeigt, an welcher Stelle im Quelltext diese Methode eingebunden wird. Die Variable *wait_maximum()* wird hauptsächlich in *reference_move_position_xy* verwendet.

Bei der ausgelegten Kommunikation sendet der Chuck zu jedem empfangenen Kommando einen Bestätigungsstring. So wird der Chuck, nach [28], auf das Kommando *MP D X -2.345 Y .1234* mit dem String *RR MP D X Y* antworten, sobald der Fahrbefehl aus dem Kommando ausgeführt ist. Sollte der Chuck den Fahrbefehl nicht ausführen können,

da die Position im Koordinatensystem nicht erreichbar ist oder dieser durch eine andere Störung bei der Ausführung des Befehls unterbrochen wird, versendet dieser den Antwortstring nicht. Mit der Methode *reference_move_position_xy* soll der zu erwartende Befehl ausgelesen und die zuvor beschriebenen Fehlerfälle abgefangen werden.

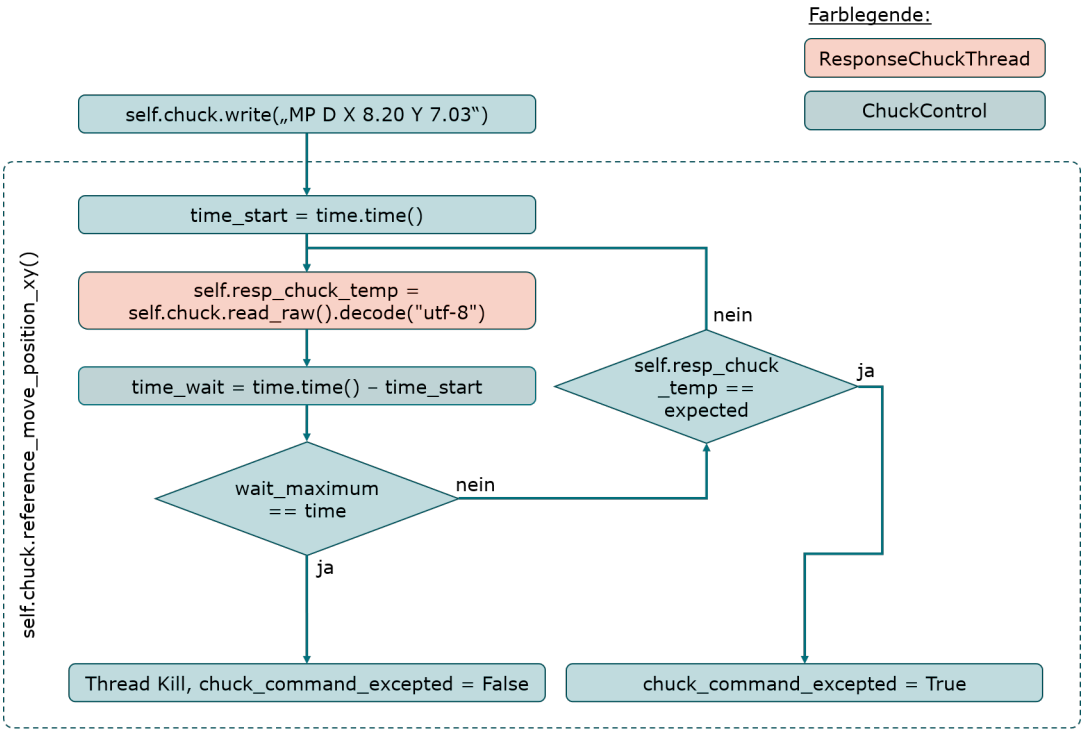

Abbildung 5.5: Ablaufdiagramm der Methode *reference_move_position_xy*

In Abbildung 5.5 sind zwei verschiedene Farbdarstellungen sichtbar, welche in der Legende oben rechts beschrieben sind. In grün sind die Aktionen dargestellt, die in der Klasse *ChuckControl* realisiert sind. Zum Abfangen der Fehlerquellen wurde zusätzlich die Klasse *ResponseChuckThread* entwickelt. Die Aktionen zu dieser Klasse sind in orange gekennzeichnet. Außerdem ist zu erwähnen, dass der gestrichelte Bereich den Inhalt der Methode *reference_move_position_xy* darstellt.

Beginnend wird außerhalb der Methode *reference_move_position_xy* das Kommando für eine Bewegung in X- und Y-Achse auf die Position 8.2/7.03 gesendet. Sobald der Chuck dieses Kommando erhalten hat, wird dieser den Fahrbefehl ausführen. In der Methode *reference_move_position_xy* wird jetzt die Startzeit in *time_start* festgehalten. Des Weiteren wird über *pyvisa* der Befehl *read_raw* ausgeführt und auf die Antwort des

Chucks gewartet. Damit das Lesen über VISA unterbrochen werden kann, wird dieses über den neuen Thread in der Klasse *ResponseChuckThread* ausgeführt, welcher nach der *maximum_time* terminiert wird. Sollte im Thread die Antwort des Chucks vor Ende der *maximum_time* gelesen werden, wird der Thread normal beendet. Ist dies der Fall, wurde das Kommando empfangen und ausgeführt. Sollte der Thread terminiert worden sein, muss das Kommando erneut gesendet werden.

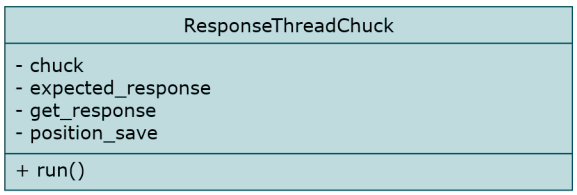

Abbildung 5.6: Klassendiagramm der Klasse *ResponseThreadChuck* zum Modul *chuck_hardware.py*

In Abbildung 5.6 ist das Klassendiagramm zu *ResponseChuckThread* zu finden. Abschließend ist hinzuzufügen, dass die Thread-Verwaltung im gesamten Programm über die Bibliothek *PyQt* und die darin enthaltende Klasse *QThread* durchgeführt wird. So wird auch das Threading der Objekte von *ResponseThreadChuck* über *QThread* verwaltet.

In der Klasse *ChuckControl* können die Methoden in drei verschiedene Typen unterteilt werden, wie es auch schon in Tabelle 5.2 angedeutet ist. Hierbei existieren die Bewegungsbefehle, die Abfragen und die Einstellungsbefehle. Auf jedes dieser Kommandos folgt der bereits beschriebene Ablauf aus Abbildung 5.5. Im Gesamtprogramm findet das Modul *chuck_hardware.py* in mehreren Bereichen Anwendung, zu denen im Folgenden die einzelnen Ablaufdiagramme vorgestellt werden.

5.2.3 Kontaktierung des Wafers über Stepping

In Abbildung 5.7 ist der Ablauf für das Stepping über den Wafer abgebildet. Dort sind die Aktionen mit grauem Text wieder anderen Modulen als *chuck_hardware.py* zugeordnet.

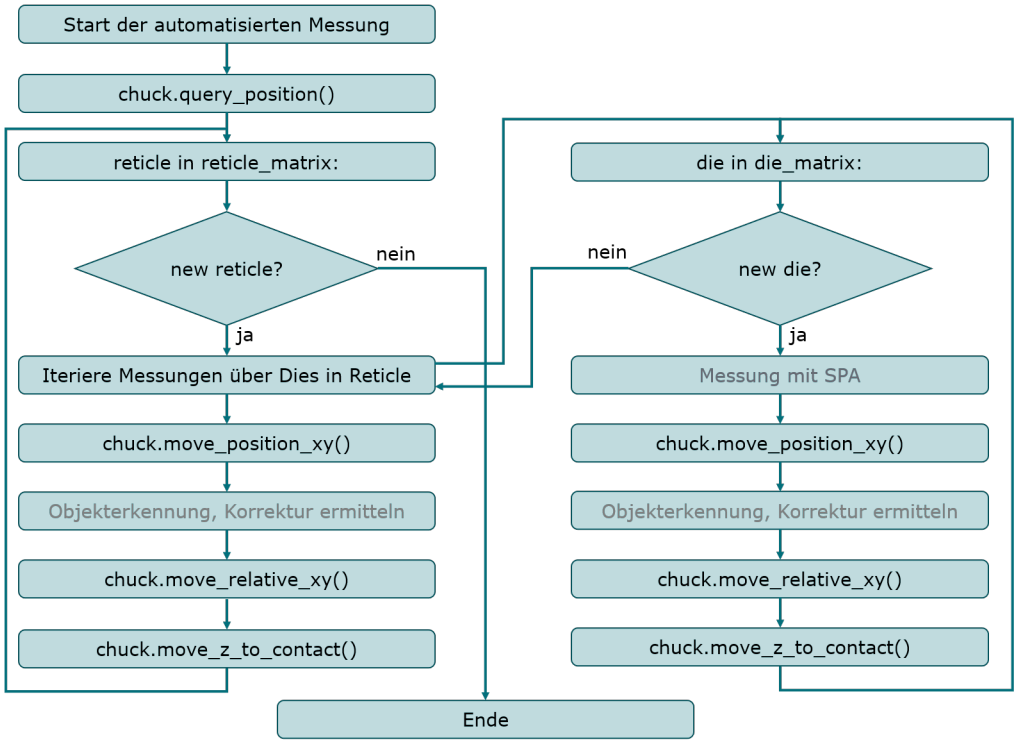

Abbildung 5.7: Ablaufdiagramm zum Stepping mit dem Chuck inklusive der Korrektur über Objekterkennung

Im Gesamtprogramm wird der*die Nutzer*in die zu messenden DUT und Reticles auswählen. Außerdem wird er den X- und Y-Pitch des DUT und des Reticles zum Wafer konfigurieren. Nachdem der*die Nutzer*in diese Einstellungen vorgenommen, die Kontaktierung des ersten DUT durchgeführt und die automatisierte Messung gestartet hat, wird die aktuelle Position des Chucks mit *query_position* abgefragt. Diese ist als zweite Aktion in Abbildung 5.7 zu finden. Auf Basis dieser Position werden im Folgenden die zu kontaktierenden Positionen über den eingestellten Reticle- und DUT-Pitch berechnet. Nach der Positionsabfrage werden im ersten Reticle am bereits kontaktierten DUT alle Messungen mit dem SPA durchgeführt. Dieser Punkt ist in Abbildung 5.7 rechts zu sehen. Hierbei wird das bereits beschriebene Ablaufdiagramm aus Abbildung 5.3 durchlaufen, sodass am kontaktierten DUT alle ausgewählten Messungen durchgeführt werden. Darauffolgend wird das nächste DUT im Reticle kontaktiert, welches im Messauftrag bearbeitet werden soll. Über den Befehl *move_position_xy* wird die Position des DUT angefahren. Weitergehend wird über die Objekterkennung festgestellt, wie weit der Chuck in X- und Y-Richtung verschoben werden muss, damit das DUT exakt unter den

Messspitzen positioniert wird. Diese kalkulierte Korrektur, welche unter Abschnitt 6.4 detallierter beschrieben wird, nutzt die Funktion *move_relative_xy* zur abschließenden Korrekturfahrt. Beendet wird diese Iteration über den Befehl *move_z_to_contact*, wobei das DUT mit den Messspitzen kontaktiert wird. So werden in dem rechten Bereich von Abbildung 5.7 die DUT im Reticle abgearbeitet. Darauffolgend wird das nächste Reticle kontaktiert, wobei ebenfalls die Position aus den eingegebenen Daten berechnet, über *move_position_xy* angefahren und mit *move_relative_xy* korrigiert wird. Auch beim Reticle wird die Iteration über die Kontaktierung mit dem Befehl *move_z_to_contact* abgeschlossen. Beendet ist der gesamte Ablauf nach Abarbeitung der ausgewählten DUT in den selektierten Reticles.

5.2.4 Kontaktierung vereinzelter DUT

Ein weiterer Ablauf ist die automatisierte Kontaktierung der vereinzelten DUT. Hierzu wird im Folgenden das Ablaufdiagramm vorgestellt.

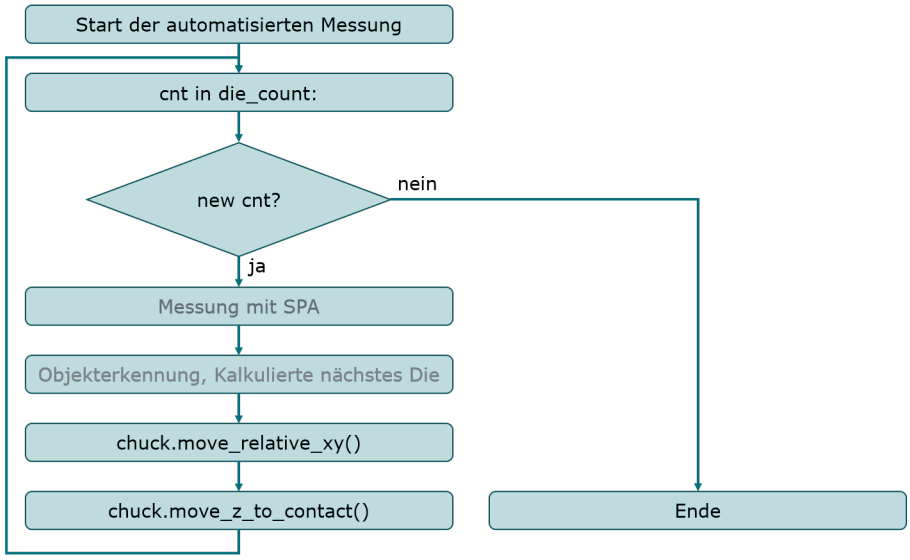

Abbildung 5.8: Ablaufdiagramm der automatisierten Kontaktierung von vereinzelten DUT

Ähnlich wie beim Stepping werden zu Beginn alle Messungen am ersten kontaktierten DUT durchgeführt. Mit der Objekterkennung wird das nächste DUT im Bild erkannt.

Hierzu wird die neue Position kalkuliert, welche über den Befehl *move_relative_xy* angefahren wird. Anders als beim Stepping, kann in diesem Aufbau auf die aktuelle Positionsabfrage verzichtet werden, da hier lediglich über die relative Fahrt die Kontaktierung durchgeführt wird. Abgeschlossen wird diese Iteration ebenfalls über den Befehl *move_z_to_contact*. Wie oft diese Iteration durchgeführt wird, hängt von der eingegebenen Anzahl an vereinzelten DUT ab.

5.2.5 Initialisierung des Chucks

Die Hardware zu diesem Modul wird neben der Automatisierung auch manuell bedient. Die händische Messung ist für einen Messauftrag mit zwei DUT deutlich schneller als die automatisierte Messung, die vor allem für größere Messaufträge entwickelt wurde. Dies kann dazu führen, dass initiale Parameter der Hardware umkonfiguriert werden. Beispielsweise wird die Hubhöhe des Chucks für DUT im Gehäuse umgestellt. Für diesen Fall wird eine Initialisierungsroutine implementiert, wobei einzelne Einstellungsbefehle zum Chuck gesendet werden.

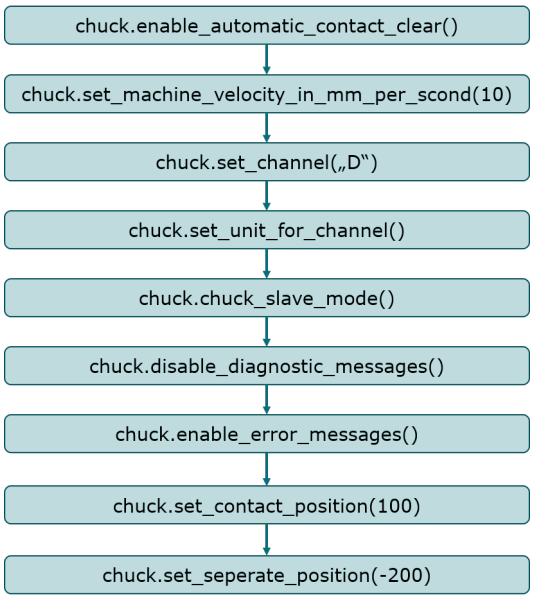

Abbildung 5.9: Ablaufdiagramm der Initialisierung des Chucks

Sollte der Chuck einen Fahrbefehl ohne die Einstellung der Z-Achse erhalten, während sich dieser in der Kontaktierungsebene befindet, werden die positionierten Messspitzen

über den Wafer kratzen und diesen beschädigen. Daher wird über *enable_automatic_contact_clear* beginnend die Bewegung der Z-Achse parametriert. Über diesen Befehl fährt der Chuck in die separate Ebene sobald ein Bewegungsbefehl zum Chuck gesendet wird. Mit diesem Kommando wird somit einer Beschädigung des Wafers vorgebeugt. Des Weiteren werden Einheiten eingestellt und hardwarespezifische Einstellungen vorgenommen. Außerdem wird die Kommunikation zwischen der Automatisierung und dem Chuck parametriert, wobei die Diagnosenachrichten deaktiviert und die Errornachrichten aktiviert werden. Abschließend wird jeweils eine feste Position für die Kontaktierungs- und separate Ebene festgelegt.

5.3 Kamerasteuerung

Dieses Modul wird in *camera_hardware.py* umgesetzt. Inbegriffen ist die Klasse *BaslerCamera* die im folgenden Klassendiagramm vorgestellt wird.

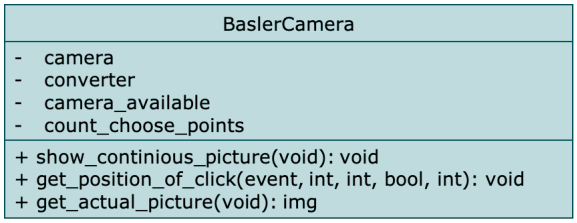

Abbildung 5.10: Klassendiagramm zum Modul *camera_hardware.py*

Mit der Bibliothek *pypylon* wird ein Interface zur Hardware bereitgestellt, welches in diesem Modul Anwendung findet. Die geöffnete Schnittstelle wird in der Objektvariable *camera* abgelegt. Mit dem Attribut *converter* bieten sich mehrere Konvertierungsmethoden, welche auf das Bild angewendet werden können. Beispielsweise wird *converter* direkt beim Einlesen des Bildes verwendet, um das Bild auf den Typ *BRG8packed*[2] umzuwandeln. Die Klasse *BaslerCamera* bietet drei Methoden, wobei die Methode *show_continious_picture* lediglich für die Inbetriebnahme des Moduls *chuck_hardware.py* entwickelt wurde. Somit können Bewegungsbefehle für den Chuck optisch kontrolliert werden. Die Methode *get_actual_picture* findet vor allem dann Anwendung, wenn die Objekterkennung, welche in Kapitel 6 erläutert wird, zur Korrektur oder Positionserfassung verwendet wird. Mit der Methode *get_position_of_click* wird in der Konfigurationsphase der

[2]Spezielles Bildformat: 8-Bit blau-rot-grün packed Format, für mehr Informationen sei auf das Kommando help(pylon.ImageFormatConverter()) in der IPython-Konsole verwiesen.

Automatisierung der Kontaktierungspunkt im Bild durch den*die Nutzer*in festgelegt. Dieser Referenzpunkt bildet die Grundlage für die Korrekturermittlung für Wafer und die Positionsermittlung für vereinzelte DUT.

5.4 Lichtsteuerung

Die Lichtsteuerung wird mit der Klasse *CameraLight* in dem Modul *camera_light_hardware.py* realisiert. Im Folgenden wird das Klassendiagramm vorgestellt.

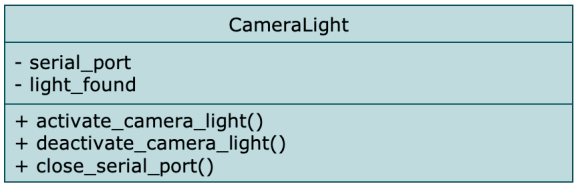

Abbildung 5.11: Klassendiagramm zum Modul *camera_light_hardware.py*

Das Attribut *serial_port* konfiguriert die Hardwareschnittstelle, über die der Schaltbefehl gesendet wird. Über die Methoden *activate_camera_light()* und *deactivate_camera_light()* wird das Licht an- und ausgeschaltet. Für das Positionieren der Messspitzen wird über *activate_camera_light()* das Licht eingeschaltet. Im Automatisierungsablauf wird das Licht beim Anwenden der Objekterkennung angeschaltet. Für die Messung des DUT muss die Lichtquelle ausgeschaltet sein, damit die Messergebnisse nicht verfälscht werden.

5.5 Benutzerschnittstelle - GUI

In dem Modul *main_darkboxprober_application.py* wird die entwickelte Benutzerschnittstelle für die Automatisierung des Darkboxprobers realisiert. Umgesetzt wird dieses Modul mit der GUI Bibliothek *PyQt5*. Neben der Benutzerschnittstelle wird in diesem Modul das Hauptprogramm umgesetzt, wobei die einzelnen, bisher beschriebenen Module verknüpft werden. Beginnend wird die GUI auf der Ebene von dem*der Nutzer*in vorgestellt. Anschließend wird dann die softwareseitige Struktur beschrieben, wobei weiterhin Klassendiagramme und Ablaufdiagramme als Beschreibungsformat verwendet werden. Für den Quelltext sei ebenfalls auf die DVD im Anhang dieser Masterthesis verwiesen.

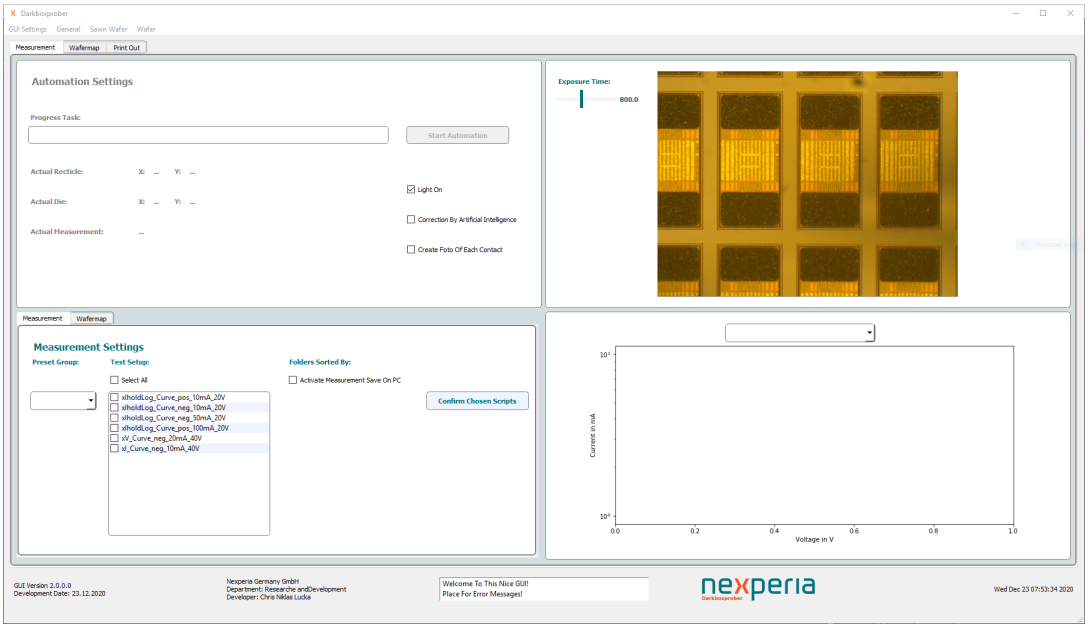

Abbildung 5.12: Die GUI nach dem Start

In Abbildung 5.12 ist die entwickelte GUI zu sehen. Diese beinhaltet die Menüleiste im oberen Bereich, den inneren Bereich mit einer Registerauswahl und die Statusleiste im unteren Bereich. Über die Statusleiste werden dem*der Nutzer*in Informationen wie die Versionsnummer und das Entwicklungsdatum der GUI bereitgestellt. Außerdem wird die Uhrzeit angezeigt, wodurch die Aktivität der GUI im laufenden Betrieb widergespiegelt wird.

5.5.1 Menüleiste

Über die Menüleiste kann der*die Nutzer*in verschiedene Aktionen anstoßen. In der folgenden Abbildung werden die einzelnen Menüs der Leiste aufgezeigt, wobei jeweils die Aktionen zu sehen sind.

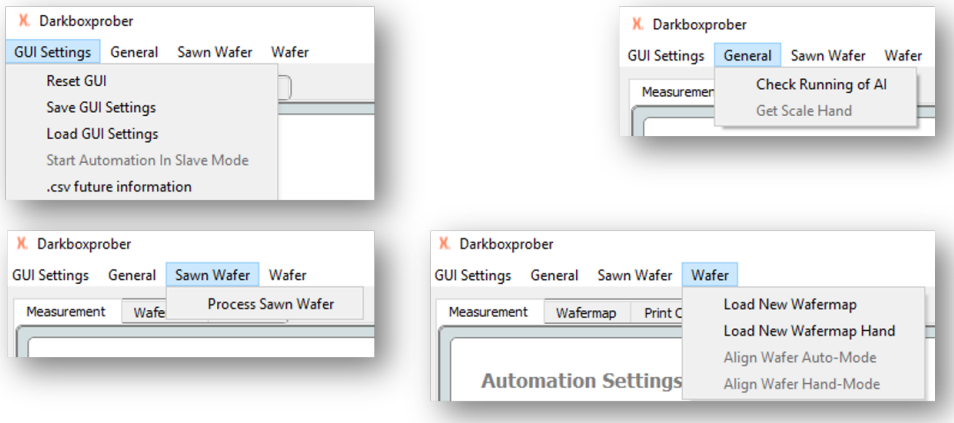

Abbildung 5.13: Die Menüleiste der GUI

Wie in Abbildung 5.13 zu erkennen ist, bieten die vier Menüs unterschiedliche Aktionen.

Menüpunkt *GUI Settings*

Über das Menü *GUI Settings* können allgemeine Einstellungen in der GUI vorgenommen werden. Die bereits eingegebenen Daten in der GUI können über *Reset GUI* gelöscht werden, wodurch sich diese wieder im initialen Zustand befindet. Über die Aktion *Save GUI Settings* können die bereits eingegebenen Daten gesichert werden und zu einem anderen Zeitpunkt über *Load GUI Settings* wieder in die GUI geladen werden. Außerdem kann die GUI aus diesem Menü über die Aktion *Start Automation In Slave Mode* in den Slavemode umgeschaltet werden.

Menüpunkt *General*

In dem Menü *General* kann das Skalieren gestartet werden. Hierbei wird die Skalierung zwischen dem Koordinatensystem des Chucks und dem Kamerakoordinatensystem ermittelt, indem der*die Nutzer*in im Kamerabild die Abmaße eines DUT markiert und diese ins Verhältnis zu dem Koordinatensystem des Chucks gebracht werden. Außerdem kann in diesem Menü über *Check Running of AI* die Funktionalität der Objekterkennung optisch überprüft werden.

Menüpunkt *Sawn Wafer*

Mit der Aktion *Process Sawn Wafer* wird ein Eingabefeld eröffnet, indem die Daten zu den vereinzelten DUT eingegeben werden können.

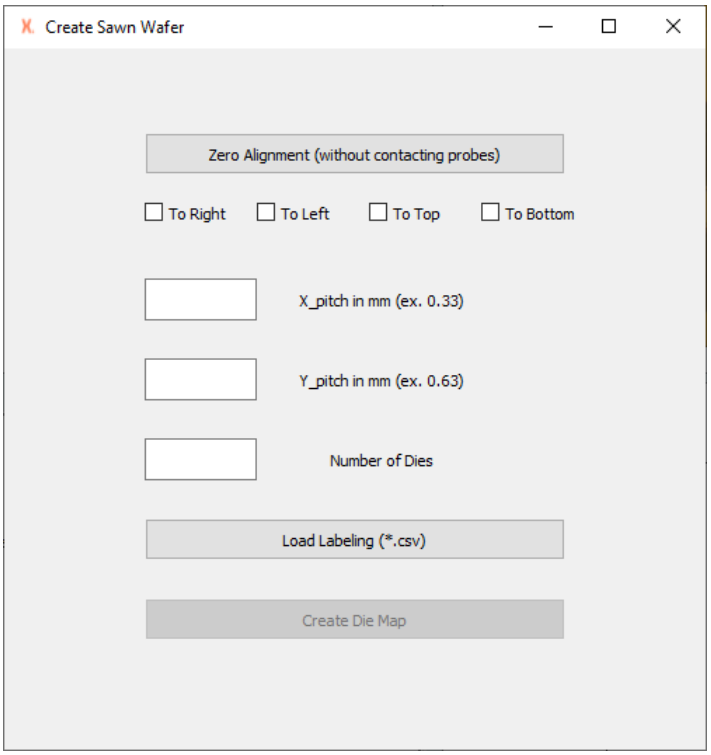

Abbildung 5.14: Das Eingabefenster *Process Sawn Wafer*

Mit diesem Eingabefeld gibt der*die Nutzer*in den Pitch des DUT ein. Außerdem wird ein Dokument über *Load Labeling* in die GUI geladen, in dem die Benennung der DUT zu finden ist. Des Weiteren wird die Anzahl der positionierten DUT angegeben. Nach Eingabe aller Daten kann über *Create Die Map* die Karte für die vereinzelten DUT in der GUI erstellt werden.

Menüpunkt *Wafer*

Über *Load New Wafermap* werden zwei verschiedene Dateiformate in die GUI geladen. Aus diesen Dateien werden Informationen über den vorliegenden Wafer verwendet, um

eine Wafermap zu erzeugen. Der Aktionspunkt *Load New Wafermap Hand* erzeugt ähnlich wie in Abschnitt 5.5.1 ein Eingabefenster zur händischen Eingabe der Daten zum Wafer und bildet eine Alternative zu der Aktion *Load New Wafermap*.

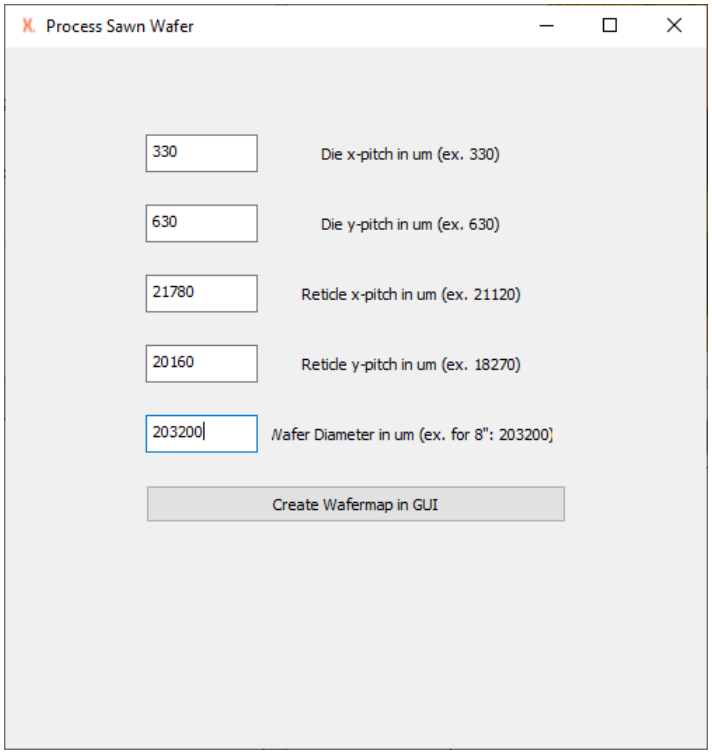

Abbildung 5.15: Das Eingabefenster *Load New Wafermap Hand*

Mit dem Button *Create Wafermap in GUI* wird das Erstellen der Wafermap aus den eingegebenen Daten gestartet. Das Menü beinhaltet außerdem die Aktion *Align Wafer Hand-Mode*, wobei das Ausrichten des Wafers gestartet wird. Hierfür wird die Interaktion durch den*die Nutzer*in gefordert. Zu detaillierteren Informationen zu dieser Funktionalität sei auf die Arbeitsanweisung zur Software hingewiesen[3].

[3]Die Arbeitsanweisung ist auf der beigefügten DVD zur Masterthesis in *Masterthesis:/97_Dokumentation/03_Arbeitsanweisung_Bedienung/* zu finden.

5.5.2 Registerauswahl Measurement

Nach dem Start der GUI ist immer das Register *Measurement* geöffnet, wie es auch in Abbildung 5.12 zu erkennen ist. Dort finden sich 4 verschiedene Teilbereiche, die im Folgenden vorgestellt werden.

Teilbereich *Automation Screen*

Abbildung 5.16: Der *Automation Screen* in der GUI

Im *Automation Screen* befindet sich der Button zum Starten der verschiedenen Anwendungen. Dabei wird anhand der eingegebenen Daten in der GUI entschieden, welche Applikation durchgeführt werden soll. Hat der*die Nutzer*in beispielsweise Daten zu vereinzelten DUT eingegeben, wird beim Klicken von *Start Automation* die Applikation vereinzelte DUT ausgeführt. In dem Screen kann vor dem Start der Applikation angewählt werden, ob die Objekterkennung eingesetzt wird, oder ob das Stepping ohne Korrektur durchgeführt werden soll. Auch das Licht kann für diese Anwendung ausgeschaltet werden. Über die Fortschrittsanzeige wird visuell dargestellt, wie weit der Messauftrag abgearbeitet ist. Die Textfelder unter der Fortschrittsanzeige beinhalten die aktuellen Positionsdaten des Chucks in der Wafermap. Außerdem wird das aktuell ausgeführte Messskript des SPAs angezeigt.

Teilbereich *Measurement Screen*

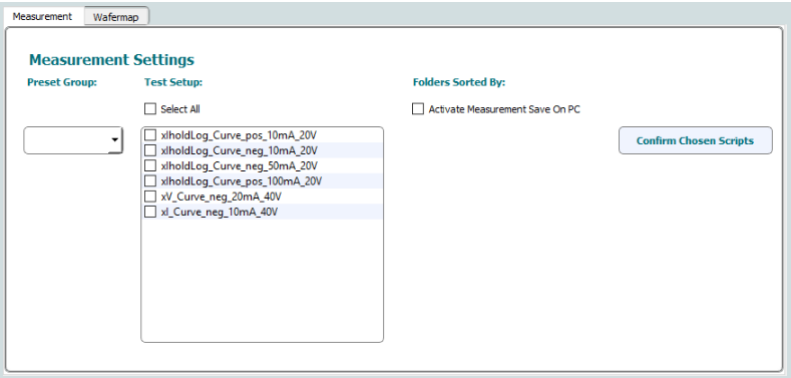

Abbildung 5.17: Der *Measurement Screen* in der GUI

Der *Measurement Screen* stellt die direkte Schnittstelle zum SPA. dar Hier kann über das Drop-Down Menü die *Preset Group* ausgewählt werden. Nach Auswahl der Gruppe werden die zugehörigen Messskripte automatisch angefordert und in die Liste des *Test Setups* eingetragen. Der*die Nutzer*in kann dort jede Messung, die durchgeführt werden soll, über eine Checkbox auswählen. Über den Button *Confirm Chosen Scrips* bestätigt der*die Nutzer*in seine Auswahl. Über die Checkbox *Activate Measurements Save On PC* wird dem*der Nutzer*in eine Auswahl an Speichermöglichkeiten angeboten, wobei die erstellte Ordnerstruktur zur Ablage der Daten, je nach getätigter Auswahl, angepasst wird.

Neben den eben beschriebenen *Measurement Settings*, ist in diesem Teilbereich auch die Registerkarte Wafermap zu finden. Hier wird über eine visuelle Darstellung der Fortschritt der Messung über dem Wafer bereitgestellt.

Teilbereich *Camera Screen*

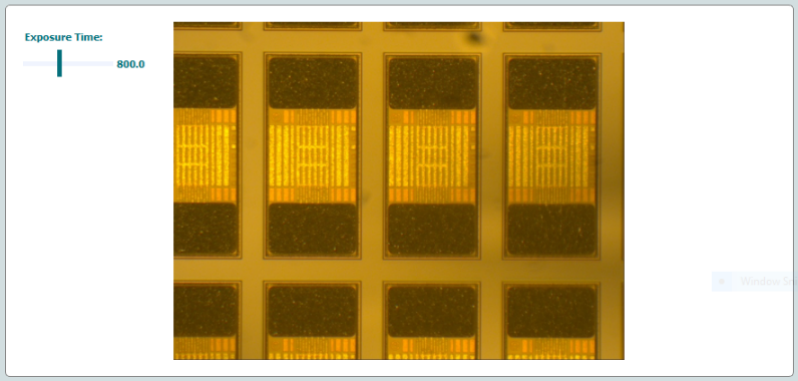

Abbildung 5.18: Der *Camera Screen* in der GUI

Im *Camera Screen* wird dem*der Nutzer*in das aktuelle Bild der Kamera angezeigt. Außerdem kann der*die Nutzer*in die Belichtungszeit vor dem Start der Messung über den Schieberegler *Exposure Time* einstellen.

Teilbereich *Plot Screen*

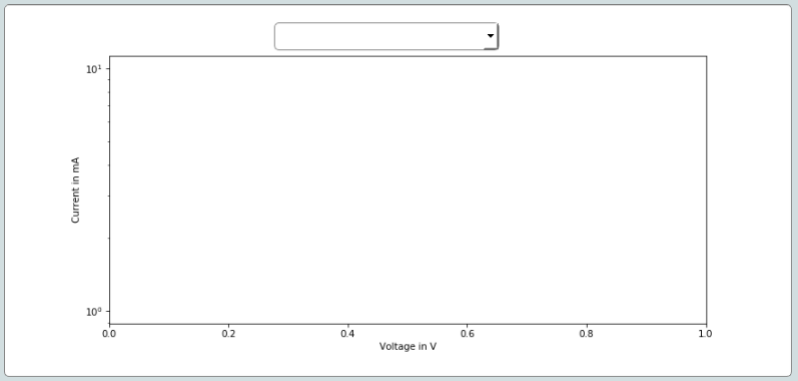

Abbildung 5.19: Der *Plot Screen* in der GUI

Je nach Auswahl der Speichermethode im *Measurement Screen*, werden dem*der Nutzer*in die Messergebnisse im *Plot Screen* angezeigt. Wird keine Messdatensicherung ausgewählt ist der Graph leer, wie es auch in Abbildung 5.19 zu erkennen ist.

5.5.3 Registerauswahl Wafermap

In der Registerauswahl *Wafermap* findet man die generierte Wafermap. Diese wird nach Eingabe der Daten zum Wafer automatisch generiert und besteht aus einer Vielzahl an Button. In der folgenden Abbildung wurden bereits exemplarisch einige Reticles und DUT ausgewählt.

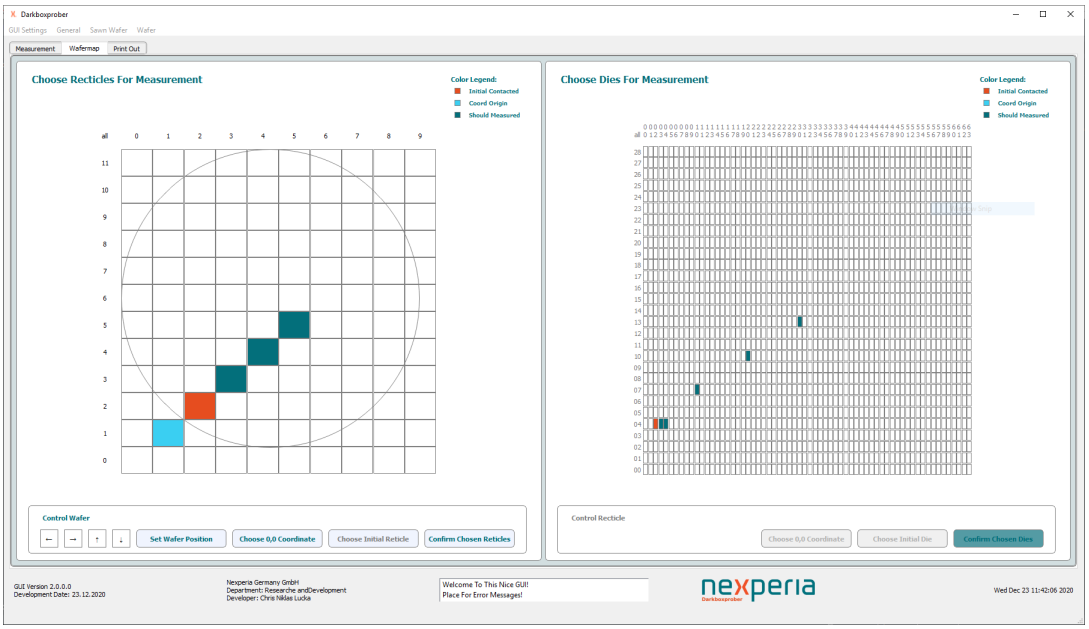

Abbildung 5.20: Registerkarte Wafermap in der GUI

In Abbildung 5.20 ist eine automatisch generierte Wafermap zu erkennen, wobei dieses Format sowohl für SPW als auf für MPW erstellt wird. Dem*der Nutzer*in werden zwei Auswahlbereiche geboten. Zum Einen die Auswahl der Reticles im linken Bereich der GUI und zum Anderen die Auswahl der DUT im Reticle im rechten Bereich der GUI.

Teilbereich *Choose Reticles For Measurement*

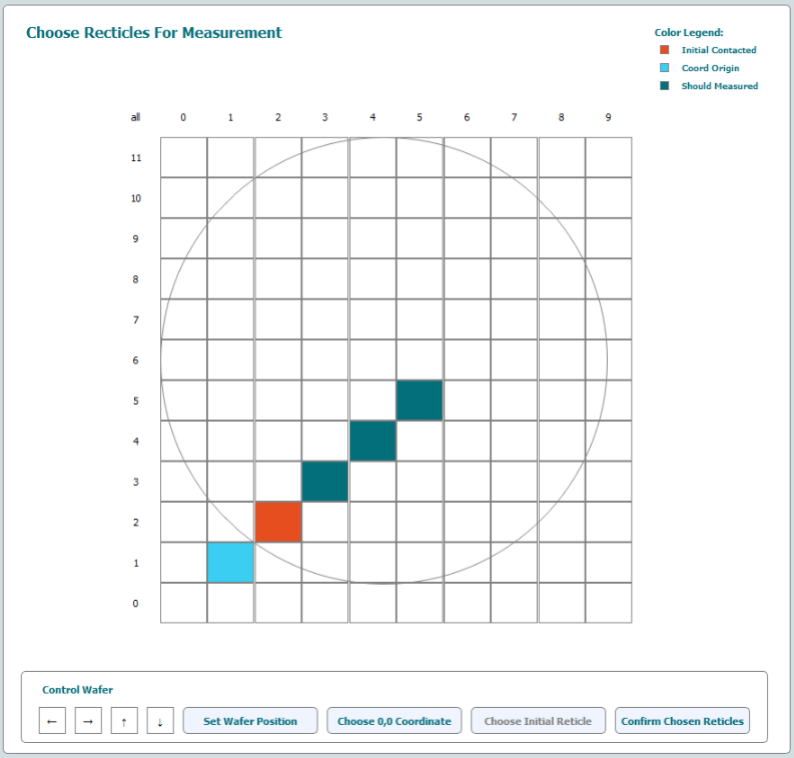

Abbildung 5.21: Automatisch generierte Wafermap zur Reticleauswahl

In Abbildung 5.21 sind verschiedene Auswahlfarben zu erkennen. In hellblau ist die Koordinate 0/0 ausgewählt. Hiermit wird der Koordinatenursprung in der Wafermap festgelegt, welcher für die Benennung der Messdaten notwendig ist. Die initiale Kontaktierung ist in orange gekennzeichnet. Dort hat der*die Nutzer*in die Messspitzen für die Messung bereits positioniert. In dunkelgrün sind die zu messenden Reticles aus dem Auftrag markiert. Über die Kontrolleinheit im unteren Bereich in Abbildung 5.21 kann der*die Nutzer*in die Auswahl bestätigen und zwischen den einzelnen Auswahlfarben umschalten.

Teilbereich *Choose Dies For Measurement*

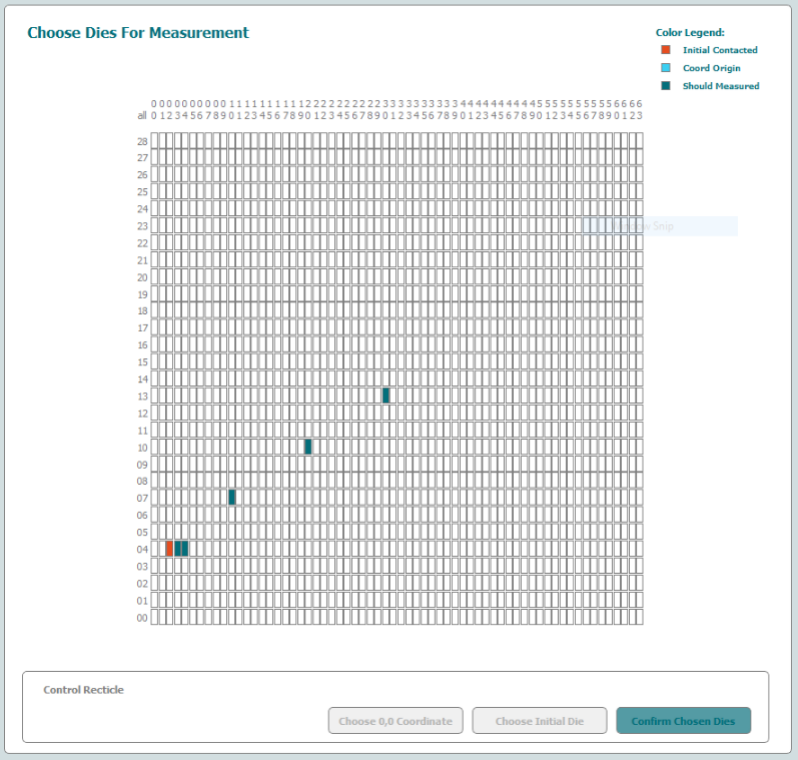

Abbildung 5.22: Automatisch generierte Reticlemap zur Auswahl der DUT

Nach dem gleichen Prinzip wie in Abschnitt 5.5.3 werden die DUT im Reticle ausgewählt. Die initiale Kontaktierung ist in orange dargestellt. In dunkelgrün sind die zu messenden DUT im Reticle zu finden. Da in diesem Beispiel keine hellblaue Markierung ausgewählt wurde, befindet sich der Koordinatenursprung des Reticles unten links in der Ecke. In jedem Reticle, welches unter Abbildung 5.21 ausgewählt wurde, werden alle ausgewählten DUT aus Abbildung 5.22 abgearbeitet. Die Auswahl muss der*die Nutzer*in vor dem Start der Messung mit dem Button *Confirm Chosen Dies* bestätigen.

5.5.4 Funktionsbeschreibung der verknüpften Module in den Applikationen

Bisher wurde die Ansteuerung der einzelnen Hardwarekomponenten realisiert und eine GUI als Schnittstelle für den*die Nutzer*in entwickelt. Außerdem ist in der GUI die Gesamtanwendung implementiert, welche die Verknüpfung der Module umfasst. Nach der Eingabe der Daten zum Messauftrag durch den*die Nutzer*in kann die Messaufgabe gestartet werden, wobei zwischen den zwei Applikationen *Wafer* oder *vereinzelte DUT* unterschieden wird. Im Folgenden wird zusammenfassend der Ablauf der jeweiligen Applikation beschrieben.

Wafer

Aus den eingegebenen Daten wird die Wafermap generiert, in der der*die Nutzer*in die zu messenden DUT und Reticles auswählt. Außerdem wird ausgewählt, welche Messskripte an jedem DUT ausgeführt werden sollen. Weitergehend muss die Skalierung durchgeführt werden, wodurch eine Relation zwischen dem Koordinatensystem des Chucks und dem der Kamera ermittelt wird. Abschließend startet der*die Nutzer*in die automatisierte Messung der ausgewählten DUT. Der weitere Ablauf wird von den Einstellungen des*der Nutzer*in bestimmt. Sollte der Haken bei *Correction by Artificial Intelligence* gesetzt sein, wird vor jeder Kontaktierung des DUT eine Korrektur über die Objekterkennung durchgeführt, sodass die Messspitzen mittig auf den Pads positioniert werden. Sollte der Haken nicht gesetzt sein, wird die Korrektur übersprungen, wodurch die Positionierung lediglich über die Koordinaten durchgeführt wird. Mit dieser Einstellung kann der*die Nutzer*in die automatisierte Kontaktierung beschleunigen. Im Betrieb ohne Korrekturfahrt kann zusätzlich das Licht ausgeschaltet werden.

Vereinzelte DUT

Bei den vereinzelten DUT wird keine Wafermap generiert, da in diesem Fall alle aufgereihten DUT im Messaufbau gemessen werden sollen. In der GUI wird direkt die Statusanzeige generiert. Der*die Nutzer*in wählt die Messskripte aus und führt die Skalierung durch. Nach dem Start der Automatisierung werden die einzelnen Messskripte ausgeführt. Weitergehend wird über einen Algorithmus und die darin verwendete Objekterkennung das nächste DUT ermittelt. Darauffolgend wird über eine relative Fahrt mit dem Chuck das

nächste DUT kontaktiert. Dieser Ablauf wird für die Anzahl der platzierten vereinzelten DUT wiederholt.

Zu den eben beschriebenen Applikationen werden die erzeugten Messdaten auf dem SPA gespeichert. Zusätzlich werden diese, je nach den Einstellungen der GUI, im Ordnerverzeichnis *Messungen* auf dem Rechner abgelegt. Die Benennung der Messdaten erfolgt nach dem eingestellten Koordinatensystem aus der Wafermap oder bei vereinzelten DUT über die geladene Liste mit Messnamen.

5.5.5 Softwarestruktur der Gesamtanwendung

Die Softwarestruktur der GUI ist in den folgenden drei Klassendiagrammen aufgezeigt. Der Ausgangspunkt *DarkboxAutomationGui* ist die Klasse, in der alle Objekte zu den Unterklassen erzeugt werden. Neben den Unterklassen, die als Benutzerschnittstelle dienen, existieren die Applikationen und die Hardwareschnittstellen.

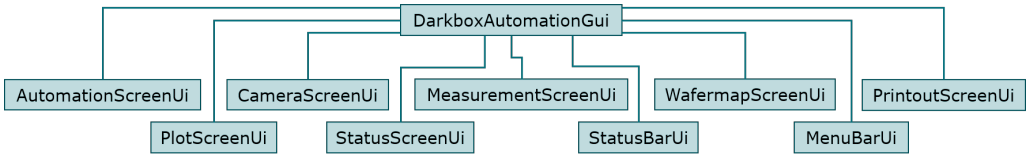

Abbildung 5.23: Softwarestruktur der GUI Teil 1

In Abbildung 5.23 sind die Unterklassen zur Realisierung der Benutzerschnittstelle zu erkennen. Die einzelnen Module, die in den vorherigen Unterkapiteln beschrieben wurden, sind hier als Unterklassen zu finden. Beispielsweise ist die Registerauswahl *Wafermap* Unterabschnitt 5.5.3 in *WafermapScreenUi* realisiert. Außerdem ist in diesem Klassendiagramm die Unterklasse *PrintoutScreenUi* zu finden, die im Vorherigen nicht beschrieben wurden. In dieser Unterklasse ist eine Konsole in der dritten Registerkarte realisiert, in der lediglich ein Protokollschreiber den abgearbeiteten Programmablauf dokumentiert und als Hilfe für den*die Nutzer*in dient.

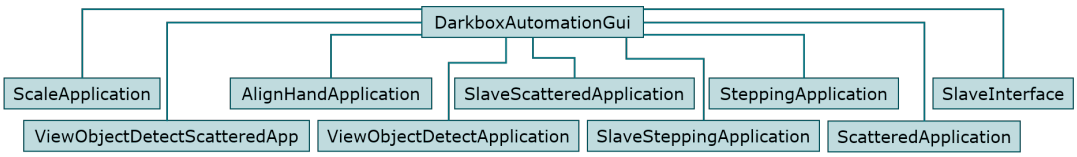

Abbildung 5.24: Softwarestruktur der GUI Teil 2

Im Klassendiagramm in Abbildung 5.24 sind die Applikationen der GUI als Unterklassen zu finden. Unter einer Applikation in der GUI versteht man beispielsweise das Ermitteln der Skalierung über die Auswahl eines DUT im Bild. Diese Anwendung ist in *ScaleApplication* realisiert. Ein anderes Beispiel ist die *SteppingApplication* Unterklasse, in der die automatisierte Messung jedes ausgewählten DUT auf dem Wafer realisiert ist. Außerdem ist in dieser Unterklasse die Korrektur über die Objekterkennung eingebunden. Die einzelnen Unterklassen nutzen somit Objekte aus der Klasse *DarkboxAutomationGui*. So werden beim Stepping die Hardwarekomponenten verwendet, welche im folgenden Klassendiagramm zu finden sind.

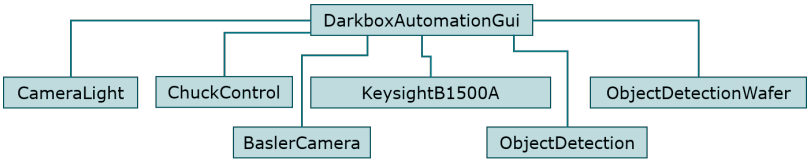

Abbildung 5.25: Softwarestruktur der GUI Teil 3

In Abbildung 5.25 sind die vier Hardwarekomponenten als Unterklassen zu finden. Auf dieser Ebene sind zusätzlich die Objekterkennungen eingeordnet. Initialisiert werden diese Objekte in der Klasse *DarkboxAutomationGui*, verwendet jedoch in den Applikationen und Benutzerschnittstellen aus Abbildung 5.23 und Abbildung 5.24.

Für detailliertere Einblicke in die Organisation des Gesamtprogramms sei auf den Quelltext auf der DVD zu der gedruckten Version dieser Masterthesis verwiesen. Insgesamt wurden in diesem Kapitel die Module zur Ansteuerung der Hardwarekomponenten umgesetzt. Außerdem wurde eine GUI entwickelt, in der die Komponenten zu einer Gesamtanwendung verknüpft wurden. Der*die Nutzer*in hat hiermit eine graphische Möglichkeit zur Bedienung dieser Anwendung. Im Folgenden wird die optische Kontrolle der Kontaktierung entwickelt, die in der Gesamtanwendung eine verlässliche Kontaktierung der Pads des DUT mit den Messspitzen umsetzen wird.

6 Erstellen der Objekterkennung mit der *Tensorflow Object Detection API*

Wie bereits in Kapitel 2 beschrieben, wird in dieser Masterthesis zur Objekterkennung die *Tensorflow Object Detection API* verwendet. Mit dieser API können vortrainierte neuronale Netzarchitekturen verwendet werden, welche auf eine spezifische Aufgabe tiefer trainiert werden können. So wird im *TensorFlow 1 Detection Model Zoo*[1] eine Vielzahl an Modellen angeboten, welche auf verschiedene Datensätze mit unterschiedlichen Inhalten vortrainiert sind. In dieser Masterthesis wird das Modell *faster_rcnn_inception_v2_pets* verwendet. Dieses soll durch die iterative Vergrößerung des Trainingsdatensatzes auf eine Genauigkeit von 70 Prozent trainiert werden. Zunächst wird die Installation der *Tensorflow Object Detection API* durchgeführt und das Training beschrieben, um darauffolgend die Trainingsergebnisse zu evaluieren. Zusätzlich wird in der Evaluation die Bewertungsform festgelegt, um die geforderte Genauigkeit messen zu können. Abschließend wird die Einbindung des Bildverarbeitungssystems in das Gesamtsystem erläutert.

6.1 Installation der *Tensorflow Object Detection API*

Für die Installation der *Tensorflow Object Detection API* sei auf die Quelle [33] verwiesen. Neben der Installationsanweisungen wird dort auch erläutert, wie das Training einer Netzarchitektur in der *Tensorflow Object Detection API* durchgeführt werden kann. Abschließend beinhaltet die Quelle die Anwendung des trainierten Netzes.

Ergänzend sei auf das Werkzeug in Quelle [36] zum Durchführen des Einrahmens und Klassifizierens von Objekten verwiesen, hierzu existieren jedoch eine Vielzahl an verschiedenen Werkzeugen.

[1] zu finden unter https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md

6.2 Training der *Tensorflow Object Detection API* auf die optische Erkennung von DUT

Für das erfolgreiche Trainieren des neuronalen Netzes muss ein Trainingsdatensatz erzeugt werden. Im Folgenden werden die einzelnen Schritte zur Erstellung dieses Datensatzes beschrieben.

6.2.1 Festlegung der Klassifikationen

Beginnend werden die Klassifikationen festgelegt. Da die Objekterkennung zum Einen auf vereinzelte DUT und zum Anderen auf Wafer angewendet werden soll, werden zwei unterschiedliche Trainingsdurchgänge durchgeführt. Im Folgenden werden die Klassifikationen zu den einzelnen Applikationen beschrieben.

Vereinzelte DUT

Bei der Bildverarbeitung in der Anwendung der vereinzelten DUT ist das Ziel, die Position des nächsten Objektes zu erhalten. Da der*die Nutzer*in die einzelnen zu messenden DUT selbstständig auf dem Chuck positioniert, übernimmt der*die Nutzer*in die Aufgabe der Objektsortierung. In diesem Fall liegen nur zu messende Objekte auf dem Chuck. Die Aufgabe der Bildverarbeitung liegt in diesem Fall in der Bestimmung der Position des nächsten DUT. Die Objekte im Bild sollen daher nach der folgenden Tabelle klassifiziert werden.

Tabelle 6.1: Festgelegte Klassifikationen für die Objekterkennung bei vereinzelten DUT

Nummer	Label	Beschreibung
1	die	realisiertes System
2	probe	Messspitzen

Über die Klassifikation *die* werden im Programm zur Ermittlung der nächsten Kontaktierungsposition die zu kontaktierenden Objekte im Bild bestimmt. Unter der Klassifikation *probe* werden die Messspitzen als Objekt im Bild markiert. Diese Klassifikation wird in

dieser Anwendung ergänzend implementiert, um weitere Informationen für Folgeanwendungen bereitstellen zu können. Außerdem kann über diese Klassifikation in Zukunft die Ermittlung des Kontaktierungspunktes im Bild automatisiert werden.

Wafer

Bei den Wafern werden weitere Klassifikationen festgelegt, da bei SPW und MPW verschiedene Strukturen auf dem Wafer vorkommen können. Unter den Strukturen sind zum Einen die bereits beschriebenen Nummerierungen des Reticles und zum Anderen die PCM-Bereiche zu verstehen, die bereits in Kapitel 2 vorgestellt wurden. Für die Korrektur einer Kontaktierung werden in dieser Masterthesis die Koordinaten der erkannten DUT im Bild verwendet. Die zugehörige Klassifikation dieser erkannten DUT wird für mögliche Folgeanwendungen implementiert. Diese kann mit der Klassifikation beispielsweise die automatische Erstellung einer Wafermap zum vorliegenden Wafer durchführen. Zur Korrektur der Kontaktierung werden die Klassifikationen jedoch nicht verwendet.

Tabelle 6.2: Festgelegte Klassifikationen für die Objekterkennung bei Wafern

Nummer	Label	Beschreibung
1	1	Zahl 1 auf Leerkörper für Reticle-Koordinatensystem
2	2	Zahl 2 auf Leerkörper für Reticle-Koordinatensystem
3	3	Zahl 3 auf Leerkörper für Reticle-Koordinatensystem
4	4	Zahl 4 auf Leerkörper für Reticle-Koordinatensystem
5	5	Zahl 5 auf Leerkörper für Reticle-Koordinatensystem
6	6	Zahl 6 auf Leerkörper für Reticle-Koordinatensystem
7	7	Zahl 7 auf Leerkörper für Reticle-Koordinatensystem
8	8	Zahl 8 auf Leerkörper für Reticle-Koordinatensystem
9	9	Zahl 9 auf Leerkörper für Reticle-Koordinatensystem
10	0	Zahl 0 auf Leerkörper für Reticle-Koordinatensystem
11	x	Buchstabe x auf einem Leerkörper
12	placeholder	Teilobjekt des PCM Feldes
13	die	Realisiertes System auf dem Wafer
14	bad_die	Realisiertes System auf dem Wafer im Randbereich
15	probe	Messspitzen

Wie bereits erwähnt, werden bei dieser Applikation mehrere Klassifikationen festgelegt, die vorerst nicht verwendet werden. Dies sind die Zahlen und Buchstaben von Nummer

1 bis 11, die in Tabelle 6.2 zu finden sind. Im Laufe des Trainings werden diese Klassifikationen jedoch als Anhaltspunkt für die Funktionalität des Bildverarbeitungssystems verwendet. Auch in dieser Applikation werden schlussendlich die Positionen der DUT verwendet, um die Positionsanfahrt zu realisieren. In diesem Fall ist die Klassifikation *probe* für Folgeanwendungen implementiert, wie beispielsweise die automatisierte Ermittlung des Kontaktierungspunktes im Bild. Mit der Klassifikation *bad_die* wird ein Platzhalter in den Klassifikationen für Folgeanwendungen implementiert. Diese Klassifikation soll zukünftig für die automatische Erkennung des Randbereiches auf dem Wafer verwendet werden. Dies ist jedoch kein Bestandteil des Trainings in dieser Masterthesis.

6.2.2 Erstellung des Trainingsdatensatzes

Wichtig bei der Erstellung des Trainingsdatensatzes ist, dass dieser die Varianz der Aufgabe abdeckt. Dabei verstehen sich unter der Varianz die verschiedenen Bedingungen, unter denen ein Bild im System aufgenommen wird. Im Folgenden werden die Punkte aufgeführt, die die Varianz des Systems beschreiben.

- Die Bildhelligkeit ist abhängig ...
  - vom Zoom des Objektivs.
  - von der Belichtungszeit der Kamera.
  - vom verwendeten Objektiv.
  - von der Reflexion des Wafers.
- Die Bildschärfe ist abhängig ...
  - vom Zoom des Objektivs.
  - vom verwendeten Objektiv.
- Die Objektgröße im Bild ist abhängig ...
  - vom Zoom des Objektivs.
  - von der realen Objektgröße.
  - vom verwendeten Objektiv.
- Die Anzahl der Objekte im Bild ist abhängig ...

    - von der Objektgröße im Bild.
    - von der realen Objektgröße.
- Die Rotation der Objekte im Bild kann unterschiedlich sein.
- Die Position der Objekte im Bild kann unterschiedlich sein.
- Die Pads des Objektes können zerkratzt sein.
- Die Objektstrukturen unterscheiden sich untereinander.
- Die Objektformen unterscheiden sich untereinander.

Über die Varianz der Anwendung ist noch einmal die Notwendigkeit eines neuronalen Netzes als Bildverarbeitungssystem zu erkennen. Die Varianz der Aufgabenstellung ist so hoch, dass ein konventioneller Bildverarbeitungsalgorithmus beliebig komplex gestaltet werden muss. In den Trainingsdatensätzen sollte diese Varianz abgedeckt werden.

Außerdem ist es wichtig, dass zu jeder Klassifikation eine große Menge an Trainingsdaten vorliegt, sodass kein Übertraining auf eine spezifische Klassifikation stattfindet. Die vorkommenden Klassifikationen müssen jedoch nicht exakt die gleiche Anzahl an Objekten in dem Trainingsdatensatz aufweisen.

Vereinzelte DUT

Im Folgenden werden einige Beispielbilder aus dem Trainingsdatensatz für vereinzelte DUT gezeigt. Bei der Erstellung des Trainingsdatensatzes war es das Ziel, die bereits beschriebene Varianz abzudecken.

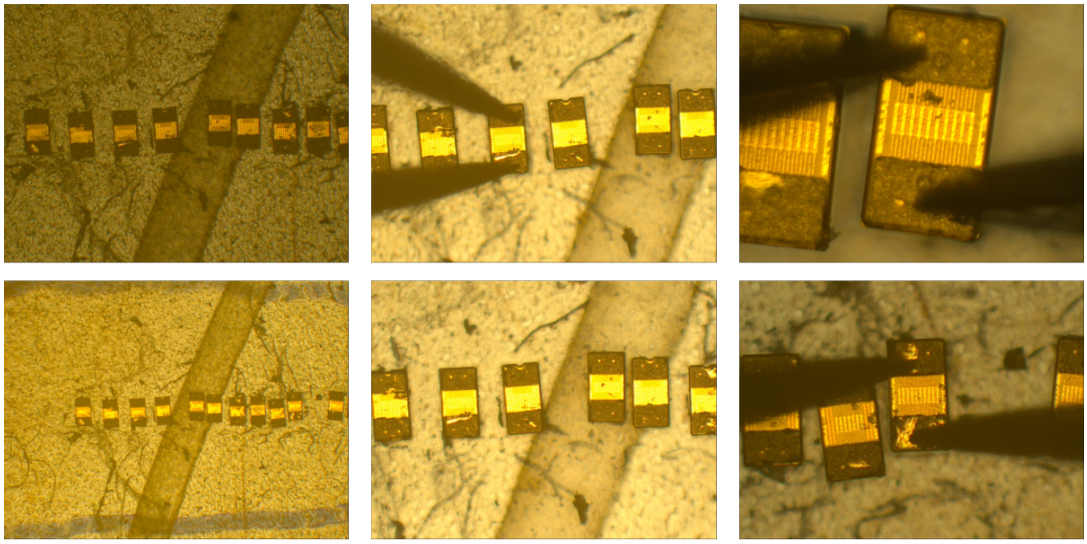

Abbildung 6.1: Exemplarische Bilder aus dem Trainingsdatensatz für vereinzelte DUT Teil 1

In Abbildung 6.1 ist von links nach rechts zu erkennen, dass unterschiedliche Objektive der Anlage verwendet wurden. Außerdem wird der Abstand des Objektivs zum Wafer so verändert, dass die DUT im Fokus des Objektivs liegen. Durch die unterschiedlichen Objektive variiert außerdem die Bildhelligkeit. Wie bei den beiden Bildern in der Mitte zu erkennen ist, wird außerdem die Objektposition im Bild verändert. So werden in diesem Beispiel im unteren Bild die Messspitzen entfernt. Außerdem wird die Position der DUT im Bild verändert, sodass auch die Varianz der Objektposition im Trainingsdatensatz abgedeckt wird. In jedem Bild ist zu erkennen, dass die Rotation der einzelnen Objekte unterschiedlich ist. Dadurch, dass die DUT aus den Bildern in Abbildung 6.1 bereits kontaktiert und/oder beschädigt wurden, ist auch die Varianz der einzelnen Objektstrukturen gegeben.

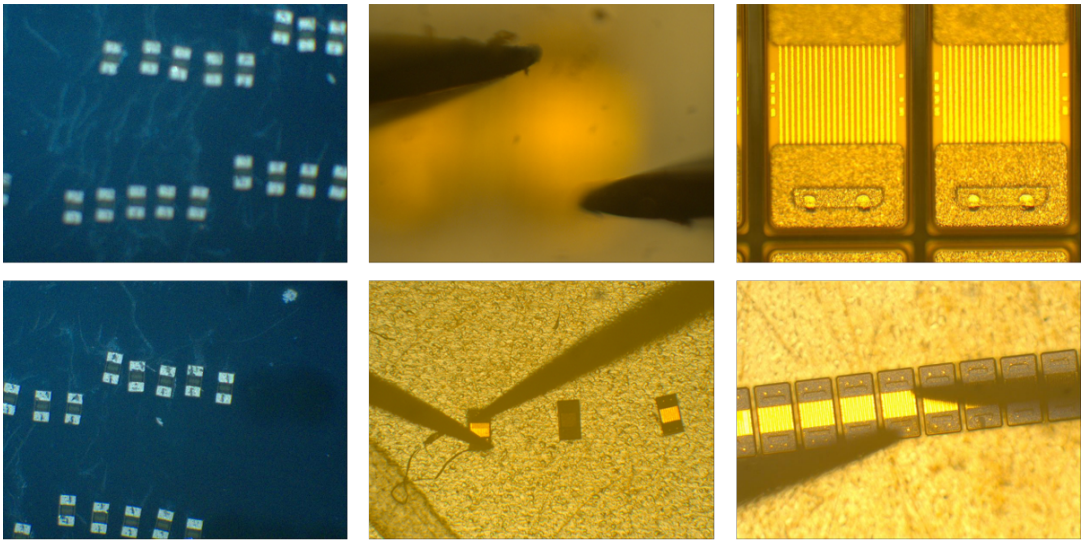

Abbildung 6.2: Exemplarische Bilder aus dem Trainingsdatensatz für vereinzelte DUT Teil 2

In Abbildung 6.2 ist auf er linken Seite die Varianz über den Fokus des Objektivs zu erkennen. Außerdem wird die Anordnung der Objekte im Bild verändert, wie es auch unten rechts zu erkennen ist. In diesem Fall wurden die Objekte sehr nah aneinander angeordnet. Die Rotation der Objekte ist in diesem Bild gleich. In der Mitte oben ist zu erkennen, dass nur die Messspitzen fokussiert werden, sodass die Anzahl der Objekte mit der Klassifikation *probe* im Trainingsdatensatz erhöht wird und der Bildverarbeitungsalgorithmus diese Klassifikation nicht nur im Zusammenhang mit der Klassifikation *die* erkennt. Oben rechts ist zu erkennen, dass auch abgeschnittene Objekte als Trainingsmaterial verwendet werden, damit die Eigenschaften der Klassifikation *die*, wie beispielsweise die sich wiederholenden Fingerstrukturen, intensiver trainiert werden können.

Des Weiteren sind in den gezeigten Beispielbildern verschiedene Systeme von insgesamt drei unterschiedlichen Wafern zu finden. Im Trainingsdatensatz finden sich weitere Systeme. Außerdem ist die Varianz aus den gezeigten Beispielbildern in den Trainingsdatensätzen höher. So wurden einige Aufnahmen mit jedem der vier verfügbaren Objektive durchgeführt. Dabei wurde jeweils eine weitere Änderung im Bild vorgenommen, sodass neben der Varianz des Objektivs auch die Varianz in der Objektanordnung realisiert wurde. Für einen detaillierten Einblick in diesen Datensatz sei auf die DVD im Anhang hingewiesen, in dem die realen Trainingsdatensätze zu finden sind.

Wafer

Im Folgenden sind einige Beispiele aus den Trainingsdaten zur Anwendung *Wafer* dargestellt. Da deutlich mehr Wafer zur Aufnahme von Trainingsbildern zur Verfügung gestellt wurden, ist die Varianz der Bilder von Wafern in dem Trainingsdatensatz deutlich höher, als die der vereinzelten DUT.

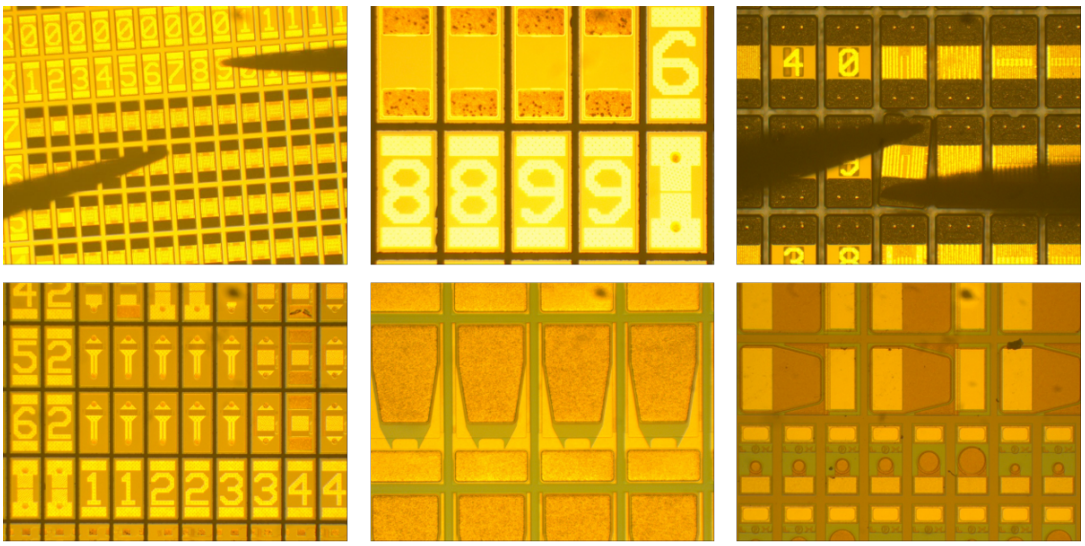

Abbildung 6.3: Exemplarische Bilder aus dem Trainingsdatensatz für Wafer Teil 1

In Abbildung 6.3 ist zu erkennen, dass die Rotation des Wafers im Datensatz als Varianz eingebunden ist. Außerdem sind die verschiedenen Systemvarianten abgedeckt. Neben den Bildern mit der Klassifikation *probe* sind auch Bilder ohne diese Klassifikation im Datensatz enthalten. Im Bild unten rechts ist zu erkennen, dass auf dem Wafer unterschiedliche Systemgrößen realisiert sind. Die Varianz durch MPW ist somit ebenfalls im Trainingsdatensatz abgebildet. In Abbildung 6.3 ist außerdem die unterschiedliche Reflexion des Wafers zu erkennen, wodurch die Bildhelligkeit variiert. In diesen Daten sind außerdem die Klassifikationen der Nummerierungen und Platzhalter zu finden. Über ein Python-Skript wurde beim Erstellen der Trainingsdaten geprüft, ob zu jeder Klassifikation eine ausreichende Anzahl an Objekten im Trainingsdatensatz vorhanden ist.

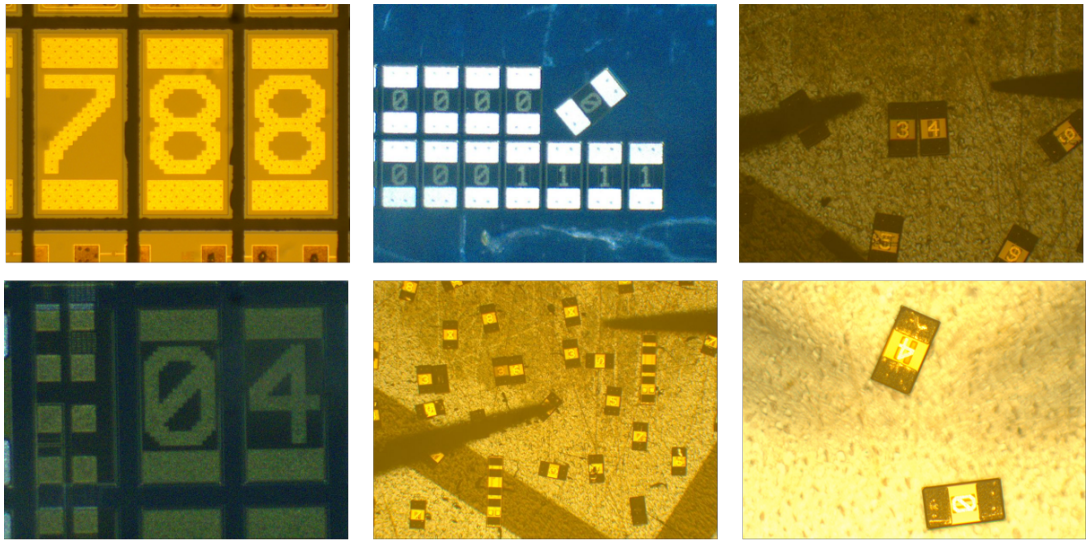

Abbildung 6.4: Exemplarische Bilder aus dem Trainingsdatensatz für Wafer Teil 2

Um die Merkmale der Nummerierungen tiefer zu trainieren, wurden von diesen Objekten separate Bilder, gelöst von dem Wafer, erstellt. Diese in Abbildung 6.4 gezeigten Bilder stellen zwar keinen realen Anwendungsfall dar, sind beim Training auf die speziellen Eigenschaften der Nummerierungen jedoch effektiv. In Abbildung 6.4 ist wieder die Varianz in der Bildhelligkeit zu finden. Außerdem werden dort wieder verschiedene Objektive mit unterschiedlicher Bildschärfe verwendet.

Die erstellten Trainingsbilder werden für die Anwendungen *Wafer* und *vereinzelte DUT* in einem Trainingsdatensatz gesammelt. So werden schlussendlich zwei Bildverarbeitungssysteme mit unterschiedlichen Klassifikationsaufgaben mit dem selben Trainingsdatensatz trainiert. In diesem Trainingsdatensatz sind zu jeder Trainingsaufgabe nur die Klassifikationen zu finden, die auch als Klassifikationsaufgabe für das neuronale Netz existieren. So sind für die Anwendung vereinzelte DUT Bilder, auf denen beispielsweise nur Nummerierungen abgebildet sind, ohne eine Klassifikation im Trainingsdatensatz abgelegt.

Labeling

Unter dem Begriff Labeling ist das händische Einrahmen und Klassifizieren von Objekten im Bild zu verstehen. Wie bereits erwähnt, wird für diese Aufgabe die Software aus [36]

verwendet. Beginnend muss sichergestellt werden, dass alle Bilder mit der Auflösung 800x600 vorliegen.

Quelltext 6.1: Kommando zum Ändern der Bildgröße aller Trainingsdaten

```
1 python resize_training_data.py -d New_folder/ -s 800 600
```

Unter der Nutzung des Moduls *resize_training_data.py* lassen sich die Bilder auf das gewünschte Format umformen. Hierfür muss *New_folder* mit dem Pfad, in dem sich die Trainingsdaten befinden, ersetzt werden. Das erwähnte Modul ist auf der beigefügten DVD unter *Masterthesis:/99_Software/07_Training/resize_training_data.py* zu finden.

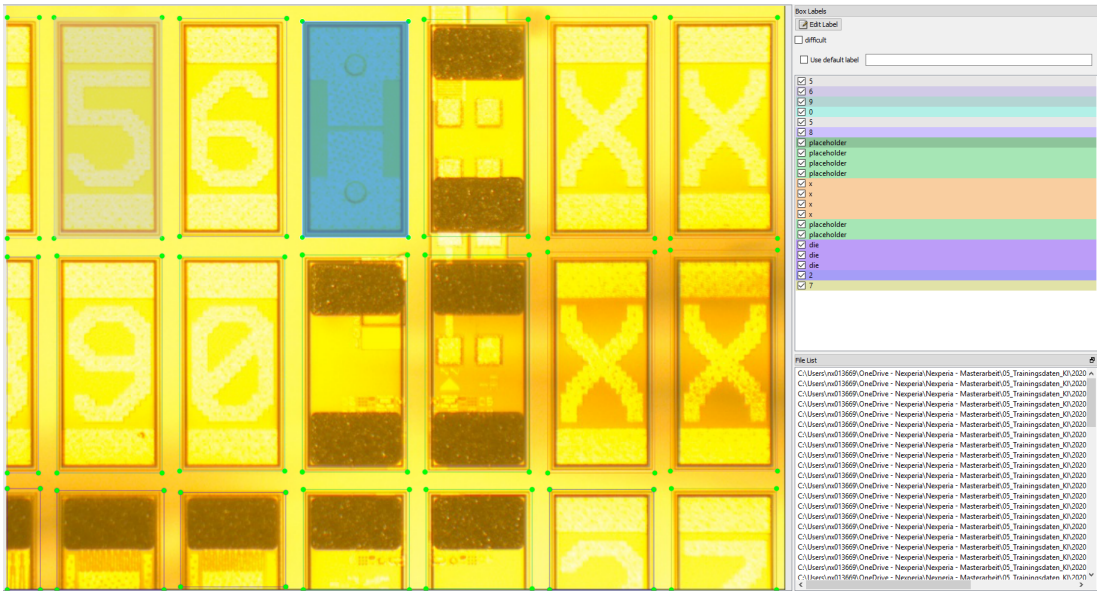

Abbildung 6.5: Exemplarisches Labeling eines Bildes von einem Wafer

So ist in Abbildung 6.5 exemplarisch ein Bild eines Wafers eingerahmt und beschriftet. Hierbei sollten sich die Kanten der einzelnen Objekte stets innerhalb des Rahmens befinden. Auf der rechten Seite im oberen Bereich sind die einzelnen Klassifikationen zu finden. Im Bild ist außerdem zu erkennen, dass alle Objekte im Randbereich des Bildes klassifiziert und eingerahmt wurden. Diese Strategie ist im gesamten Trainingsdatensatz zu finden. Nachdem das Labeling durchgeführt wurde, befindet sich zu jedem Bild eine

XML[2]-Datei im Trainingsdatensatz, in dem die einzelnen Positionen der Rahmen und die dazugehörigen Klassifikationen hinterlegt sind.

Automatische Datenerweiterung

Um nach dem Labeling die Varianz in dem Trainingsdatensatz softwareseitig zu erweitern, kann unter dem Namen *Data Augmentation*[3] der vorhandene Trainingsdatensatz ausgeweitet werden. Die Bilder werden beispielsweise automatisch gedreht, wobei das bereits existierende Label an das neue Bild automatisch angepasst wird. Im Folgenden wird ein Beispiel zur automatischen Datenerweiterung gezeigt.

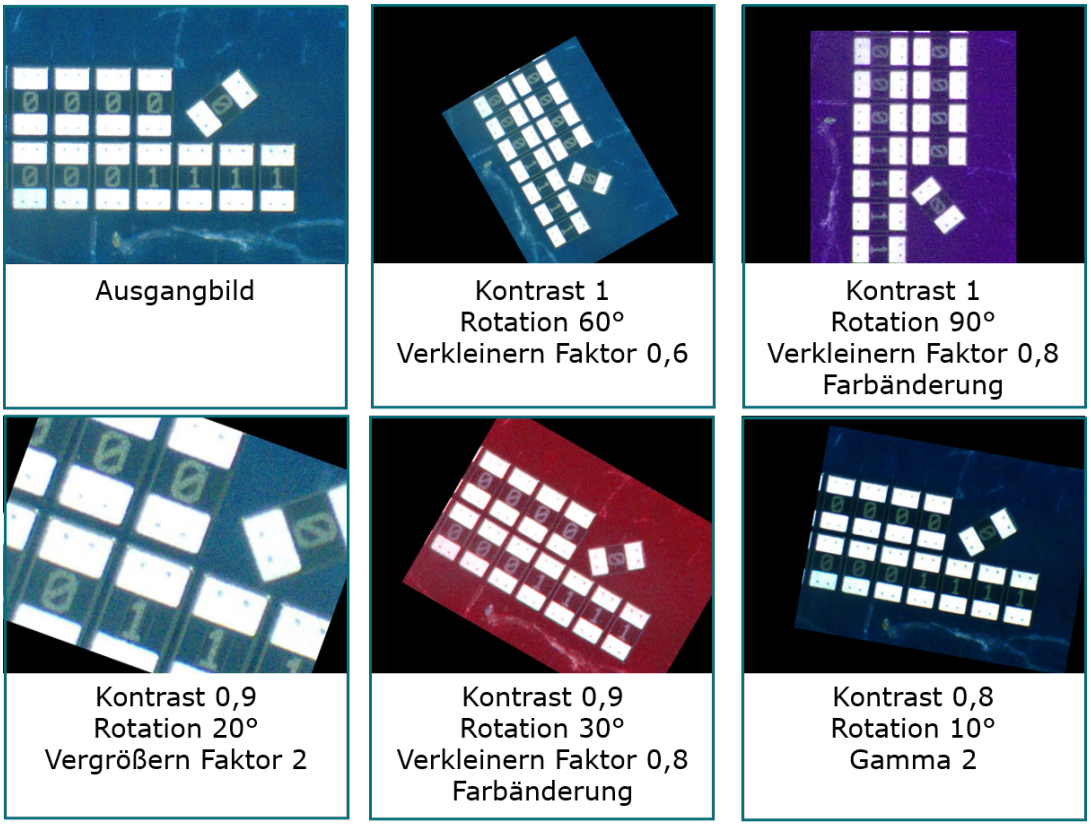

Abbildung 6.6: Automatische Datenerweiterung an einem Beispielbild

[2] **E**xtensible **M**arkup **L**anguage, englisch für erweiterbare Auszeichnungssprache
[3] englisch für Vermehrung der Daten

Durch *Data Augmentation* wurden in dieser Masterthesis, die in Abbildung 6.6 erkennbaren Erweiterungen des Trainingsdatensatzes vorgenommen, welche im Trainingsdatensatz auf der beigefügten DVD dieser Masterthesis zu einigen Bildern zu finden sind.

6.2.3 Durchführen des Trainings

Beim Training des neuronalen Netzes auf die Bildverarbeitungsaufgabe werden die Gewichte des Netzes optimiert. Für die Prüfung, ob diese Optimierung der Gewichte nach dem Trainingsdurchlauf zu einer Verbesserung der Bildverarbeitung geführt hat, wird ein Testdatensatz benötigt. Hierzu muss nach [33] der existierende Trainingsdatensatz in Trainings- und Testdaten aufgeteilt. Dabei werden 70 Prozent der Bilder in die Trainings- und 30 Prozent in die Testdaten verteilt. Damit die Varianz in den Testdaten ebenfalls hoch ist, werden diese Daten händisch ausgewählt.

Umformen der XML-Dateien zum *.record*-Format

Des Weiteren müssen die XML-Dateien, die mit in die Trainings- und Testdaten aufgeteilt wurden, in einer CSV[4]-Datei zusammengefasst werden. Über das Modul *xml_ to_ csv.py* und den folgenden Kommandos in dem Kommandofenster werden die Daten automatisch zusammengefasst.

Quelltext 6.2: Kommando zum Zusammenfassen der XML-Dateien in Trainings- und Testdaten im CSV-Format

```
1 cd \models\research\object_detection\images\test
2 python xml_to_csv.py
3 cd \models\research\object_detection\images\train
4 python xml_to_csv.py
```

Für das Training mit der *Tensorflow Object Detection API* werden die Dateien *train.-record* und *test.record* gefordert, welche über die folgenden Kommandos im Kommandofenster aus den *.csv*-Dateien generiert werden können.

[4] **C**omma-**s**epearted **V**alues, englisch für Komma-getrennte Werte

Quelltext 6.3: Kommando zur automatischen Generierung des *.record*-Formates der Trainings- und Testdaten

```
cd \models\research\object_detection
python generate_tfrecord.py
        --csv_input=images\test_labels.csv
        --image_dir=images\test --output_path=test.record
python generate_tfrecord.py
        --csv_input=images\train_labels.csv
        --image_dir=images\train --output_path=train.record
python generate_tfrecord_mpw.py
        --csv_input=images\test_labels.csv
        --image_dir=images\test --output_path=test_mpw.
           record
python generate_tfrecord_mpw.py
        --csv_input=images\train_labels.csv
        --image_dir=images\train --output_path=train_mpw.
           record
```

Da die beiden Anwendungsfälle unterschiedliche Klassifikationsaufgaben haben, werden mit dem Modul *generate_ tfrecord.py* nur die Rahmen in *train.record* und *test.record* abgelegt, die auch die bekannten Klassifikationen für die Anwendung beinahlten. So werden für die Anwendung *vereinzelte DUT* beispielsweise die Rahmen um DUT auf einem Wafer in *train.record* und *test.record* übernommen, die Rahmen der Nummerierungen jedoch nicht. Gleichermaßen werden über *generate_ tfrecord_ mpw.py* alle Klassifikationen mit in *train.record* und *test.record* übernommen, da alle Klassifikationen der Anwendung *verienzelte DUT* ebenfalls in der Anwendung *Wafer* vorkommen. So liegen abschließend vier *.record*-Dateien vor, die für das Training der neuronalen Netze verwendet werden.

Starten des Trainings

Über das folgende Kommando wird das Training der Anwendung *Wafer* mit dem bereitgestellten Trainingsdatensatz gestartet.

Quelltext 6.4: Kommando zum Starten des Trainings zur Anwendung *Wafer*

```
python train.py --logtostderr --train_dir=training/mpw/
```

```
2         --pipeline_config_path=training/mpw/
3         faster_rcnn_inception_v2_pets.config
```

Über das folgende Kommando lässt sich der Trainingsfortschritt überwachen. Hierzu muss die ausgegebene Adresse im Kommandofenster in einem Browser geöffnet werden.

Quelltext 6.5: Kommando zum Starten von Tensorboard zur Anwendung *Wafer*

```
1 tensorboard --logdir=training/mpw/ --host localhost --port
     8088
```

Die Ausgabe über die Bibliothek *Tensorboard* sieht im Browser dann wie im folgenden Bild aus.

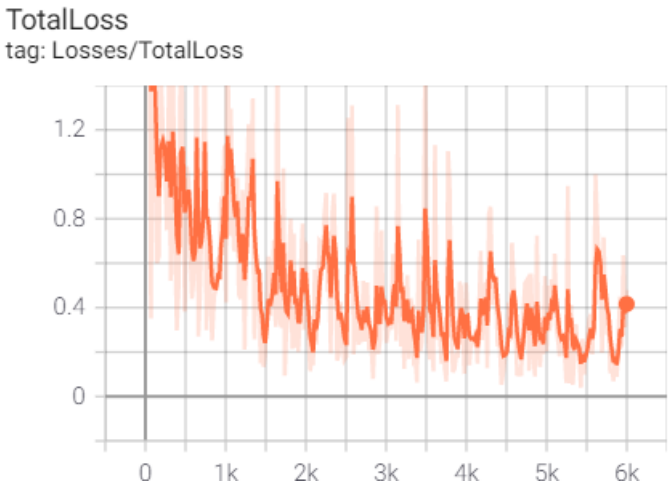

Abbildung 6.7: Trainingsüberwachung mit Tensorboard

Über den *TotalLoss*, also dem gesamten Verlust, wird bewertet, wie präzise das System mit dem aktuellen Trainingsstand die Bildverarbeitungsaufgabe löst. Hierbei werden die Testdaten als Grundlage zur Verifikation verwendet. Berechnet wird dieser Wert aus den Verlusten in der Klassifikation, Objektfindung und Lokalisation. Außerdem wird die Reproduzierbarkeit des Ergebnisses geprüft. Der gesamte Verlust liegt in Abbildung 6.7 nach 6.000 Trainingsschritten bei einem Wert von circa 0.4, wobei ein Wert von unter 1 angestrebt wurde. Wie in Abbildung 6.7 außerdem zu erkennen ist, überwiegt das Rauschen des gesamten Verlustes bereits nach 2.000 Trainingsschritten dem Ergebnis. Nach weiteren 4.000 Schritten wird das Training gestoppt. Das reale System wurde mit

70.000 Trainingsschritten optimiert, wobei das Endergebnis ähnlich wie in Abbildung 6.7 mit einem Rauschen auf dem Mittelwert von circa 0.4 stagniert.

Damit das trainierte Netz angewendet werden kann, muss dieses in ein Model umgeformt werden. Hierfür bietet die verwendete API ein Modul zur automatischen Umsetzung.

Quelltext 6.6: Kommando zum Umsetzen des Trainingsergebnisses in ein Model zur Anwendung *Wafer*

```
1 python export_inference_graph.py
2         --input_type image_tensor
3         --pipeline_config_path training/mpw/
4         faster_rcnn_inception_v2_pets.config
5         --trained_checkpoint_prefix training/mpw/model.ckpt-
             xxx
6         --output_directory inference_graph/mpw/
```

Je nach Trainingsstand muss in dem gezeigten Kommando der Name des Checkpoints *model.ckpt-xxx* angepasst werden. In dem Ordner *inference_graph* ist abschließend das Model zu finden, welches im nächsten Schritt evaluiert wird.

Die beschriebenen Anweisungen werden für die Anwendung *vereinzelte DUT* wiederholt. Auch hier wird das Training mit 70.000 Schritten durchgeführt.

6.3 Evaluation der Trainingsergebnisse

Zur Evaluation werden im folgenden die Grundlagen zur Bewertung eines Bildverarbeitungssystems vorgstellt.

6.3.1 Allgemeine Bewertung eines Bildverarbeitungssystems zur Objekterkennung

Über die folgende Tabelle ist die Genauigkeitsanalyse eines Bildverarbeitungssystems möglich. Dabei wird die aufgezeigte Tabelle auf jedes zu klassifizierende Objekt im Bild angewendet.

Klassifikation	Klassifizierungsergebnis ist **negativ**	Klassifizierungsergebnis ist **positiv**
Objekt ist tatsächlich **negativ**	true negative - TN	false positive - FP
Objekt ist tatsächlich **positiv**	false negative - FN	true positive - TP

Abbildung 6.8: Klassifikationsbewertung, modifiziert aus [9]

In Abbildung 6.8 ist die Tabelle zur Bewertung einer Klassifikation zu erkennen. Unter Verwendung eines Evaluierungsdatensatzes werden alle möglichen Objekte in jedem Bild mit dieser Tabelle geprüft. Über die Begriffe Precision und Recall kann im nächsten Schritt eine Aussage über das System getroffen werden.

Precision

Die folgende Gleichung für Precision ist aus [9] entnommen, die dazugehörige Erläuterung wurde sinngemäß übernommen.

$$Precision = \frac{TP}{TP + FP} \tag{6.1}$$

Über diese Formel werden die positiv richtig getätigten Klassifikationen ins Verhältnis zu der Gesamtheit an positiven getätigten Klassifikationen gesetzt. Somit gibt dieser Wert eine Aussage darüber, wie hoch die Wahrscheinlichkeit ist, dass eine positiv getätigte Klassifikation richtig ist.

Recall

Die folgende Gleichung für Recall ist aus [9] entnommen, die dazugehörige Erläuterung wurde sinngemäß übernommen.

$$Recall = \frac{TP}{TP + FN} \tag{6.2}$$

Für diesen Wert werden die positiv richtig getätigten Klassifikationen ins Verhältnis zu der Gesamtheit der tatsächlich positiven Klassifikationen gesetzt. Ein steigender Recall

korreliert mit der Wahrscheinlichkeit, dass ein tatsächlich positives Objekt vom Bildverarbeitungssystem richtig klassifiziert wird.

IoU - Die Schnittmenge über Vereinigungsmenge

Eine weitere wichtige Größe in der Bewertung von Objekterkennungssystemen ist die IoU[5]. Die folgende Erläuterung wurde sinngemäß aus [15] entnommen.

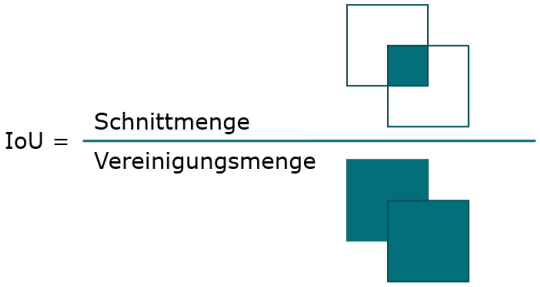

Abbildung 6.9: Visuelle Darstellung der IoU, modifiziert aus [15]

Über diese Größe wird die Schnittmenge der Fläche des Originalobjektes und des ermittelten Objektes durch das Bildverarbeitungssystem ins Verhältnis zur Fläche der Vereinigungsmenge der beiden Objekte gesetzt. Das Ergebnis ist eine Zahl zwischen Null und Eins und liefert somit eine Aussage über die Treffsicherheit der Objekterkennung.

Die Bewertung des Systems muss für jede Klassifikation einzeln durchgeführt werden. Dabei muss ein Evaluationsdatensatz verwendet werden, der keine Schnittmenge mit den Trainings- und Testdaten beinhaltet. Die *Tensorflow Object Detection API* stellt mit *model_main.py* bereits ein Evaluierungswerkzeug zur Verfügung, über das die bereits erläuterten Werte Precision und Recall über verschiedene IoU-Grenzwerte ermittelt werden.

6.3.2 Ausführen der automatischen Evaluierung

Über eine vom Training gelöste Auswertung wird mit einem neuen Datensatz im Folgenden die Funktionalität des Bildverarbeitungssystems geprüft.

[5] **I**ntersection **o**ver **U**nion, englisch für Schnittmenge über Vereinigungsmenge

Quelltext 6.7: Kommando zum Anwenden der Evalierung zur Anwendung *Wafer*

```
1 python model_main.py
2         --pipeline_config_path=training/mpw/pipeline_eval.
             config
3         --model_dir=inference_graph/mpw/saved_model/
4         --checkpoint_dir=inference_graph/mpw/
5         --run_once=True
```

In *pipeline_ eval.config* ist der Pfad zum Evaluierungsdatensatz hinterlegt. Dieser Datensatz enthält Bilder mit konkreten Anwendungsfällen für die einzelnen Bildverarbeitungsaufgaben. Im Folgenden werden die Ergebnisse der Evaluation zu den beiden Bildverarbeitungsaufgaben vorgestellt.

6.3.3 Vereinzelte DUT

Nach Anwendung des exemplarischen Kommandos aus Quelltext 6.7 auf die Anwendung *vereinzelte DUT* wird eine Bewertungsliste in der Konsole ausgegeben. Diese beinhaltet verschiedene Werte die für die spezifischen Anwendungen interpretiert werden müssen. In der folgenden Abbildung ist die Ausgabe zu der Anwendung *vereinzelte DUT* gezeigt.

```
01 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.645
02 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.929
03 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.711
04 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.008
05 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.387
06 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.702
07 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.236
08 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.542
09 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.714
10 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.067
11 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.563
12 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.759
```

Abbildung 6.10: Ergebnis der Evaluierung zur Objekterkennung von vereinzelten DUT

In Abbildung 6.10 sind in schwarzer Schrift die für die Anwendung notwendigen Kriterien zu erkennen. In grau hinterlegt sind die Kriterien zu sehen, die für diese Anwendung aussortiert wurden.

Precision

In der ersten Zeile in Abbildung 6.10 wird die Precision über eine IoU von 0.5 bis 0.95 gebildet. Außerdem wird dieser Wert zu allen erkannten Objekten unabhängig von der Objektgröße kalkuliert. Das Ziel aus dem Pflichtenheft ist das Erreichen einer Genauigkeit des Bildverarbeitungssystems von 70 Prozent. Somit reicht eine Betrachtung der Precision-Bildung unter einer IoU von 0.75, wie es in der dritten Zeile zu erkennen ist. In der zweiten Zeile wird der Wert Precision über eine IoU von 0.5 gebildet. Dieser Wert ist außerdem wichtig, da dieser zur Weiterentwicklung der Lokalisation einen weiteren Kontrollpunkt, neben der Precision bei einer IoU von 0.75, bildet. In Zeile vier bis sechs wird die Precision über eine IoU von 0.5 bis 0.95 über verschiedene Objektgrößen gebildet. Klassifiziert werden die Objektgrößen über die COCO API aus der verwendeten Evaluation. Hier finden sich unter dem Modul *cocoeval.py*[6] die folgenden vier Klassifikationen für Objektgrößen.

Tabelle 6.3: Grenzwerte der Objektgrößenklassifikationen, inhaltlich entnommen aus dem Modul *cocoeval.py* aus [35]

Klassifikation der Objektgröße	Grenzwerte in Pixel
all	0 x 0 bis 100.000 x 100.000
small	0 x 0 bis 32 x 32
medium	32 x 32 bis 96 x 96
large	96 x 96 bis 100.000 x 100.000

Im Anwendungsfall *vereinzelte DUT* sollen zwei DUT im Bild zu erkennen sein. Zum Einen das Kontaktierte, zum Anderen das nächste zu kontaktierende DUT. Mit der Optik am System wird im Anwendungsfall das DUT so groß wie möglich im Bild dargestellt, sodass diese häufig in der Objektgröße *large* klassifiziert werden. Auch die Messspitzen werden am häufigsten in der Objektgröße *large* klassifiziert werden. Demnach ist Zeile sechs in Abbildung 6.10 ein weiteres notwendiges Kriterium. Die Kriterien der Objektgrößen *small* und *medium* in Zeile vier und fünf werden nicht als Bewertungskriterien verwendet.

[6] Zu finden unter der Installation von *pycocotools*, beispielsweise auf dem Rechner zu dieser Masterthesis unter *C://Users/xxxxx/Anaconda3/Lib/site-packages/pycocotools/cocoeval.py* in Zeile 510.

So ist für die drei ausgewählten Zeilen zwei, drei und sechs jeweils ein Grenzwert von mindestens 0.7 oder höher zu erreichen, um die Genauigkeitsanforderungen von 70 Prozent zu realisieren.

Recall

In den Zeilen sieben bis neun in Abbildung 6.10 ist die Ermittlung des Recalls über eine IoU von 0.5 bis 0.95 zu finden. Außerdem wird der Recall zu verschiedenen Grenzwerten für die maximale Objektanzahl im Bild ermittelt. Da in dem Evaluierungsdatensatz Bilder vorkommen, die mehr als zehn zu erkennende Objekte aufweisen, wird lediglich die Ermittlung des Recalls mit maximal 100 Objekten in Zeile neun betrachtet. Außerdem wird bei den Zeilen zehn bis zwölf wieder zwischen unterschiedlichen Objektgrößen klassifiziert wodurch lediglich Zeile zwölf mit der Objektgröße *large* zur Ermittlung des Recalls verwendet wird. Auch an dieser Stelle ist wieder ein Wert größer gleich 0.7 gefordert, um die Anforderungen aus dem Pflichtenheft zu erfüllen.

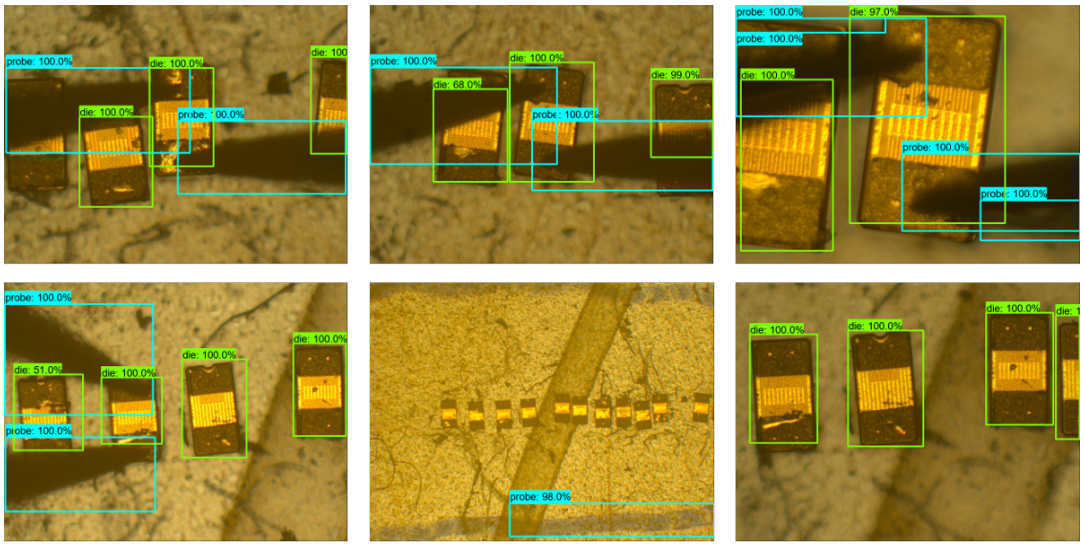

Abbildung 6.11: Anwendung der Objekterkennung für vereinzelte DUT auf Bilder im Evaluierungsdatensatz

In Abbildung 6.11 ist ein Teil der Evaluierungsdaten unter Anwendung des bereits evaluierten Bildverarbeitungssystems zu sehen. In grün eingerahmt ist die Klassifikation *die* zu finden. In hellblau eingerahmt sind die Messspitzen mit der Klassifikation *probe* zu

erkennen. In dem Bild in der Mitte unten werden die einzelnen DUT aufgrund der Objektgröße nicht mehr vom Bildverarbeitungssystem erkannt. Da die Objektgrößen *small* und *medium* aus der Bewertung genommen wurden, ist dieses ein erwartetes Ergebnis. Für die Anwendung bedeutet dieses, dass der*die Nutzer*in die Objektiveinstellungen anpassen muss. Zusätzlich ist in den Bildern zu erkennen, dass das Bildverarbeitungssystem so ausgelegt ist, dass überlappende Objekte gefunden werden können, diese jedoch nicht in der richtigen Objektgröße erkannt werden. Zukünftig muss der*die Nutzer*in die Messspitzen so anordnen, dass möglichst wenige Objektüberlappungen im Bild vorkommen, sodass die IoU optimal ausgenutzt wird. Im Bild unten rechts ist zu erkennen, wie optimal der Rahmen um das Objekt gelegt wird, wenn keine Objektüberschneidungen vorkommen.

Abschließend ist festzustellen, dass alle notwendigen Kriterien in Abbildung 6.10 die Anforderung einer Genauigkeit von 70 Prozent erfüllen. Das Training für *vereinzelte DUT* wurde somit erfolgreich durchgeführt und evaluiert.

Wafer

Nach den gleichen Kriterien der Evaluation des Anwendungsfalls der *vereinzelten Dies*, wird auch die Evaluation des Anwendungsfalls *Wafer* durchgeführt.

```
01 Average Precision (AP) @[ IoU=0.50:0.95      | area=   all    | maxDets=100 ]    = 0.588
02 Average Precision(AP) @[ IoU=0.50            | area=   all    | maxDets=100 ]    = 0.798
03 Average Precision(AP) @[ IoU=0.75            | area=   all    | maxDets=100 ]    = 0.709
04 Average Precision (AP) @[ IoU=0.50:0.95      | area= small    | maxDets=100 ]    = 0.059
05 Average Precision (AP) @[ IoU=0.50:0.95      | area=medium    | maxDets=100 ]    = 0.406
06 Average Precision(AP) @[ IoU=0.50:0.95       | area= large    | maxDets=100 ]    = 0.722
07 Average Recall     (AR) @[ IoU=0.50:0.95      | area=   all    | maxDets= 1 ]     = 0.306
08 Average Recall     (AR) @[ IoU=0.50:0.95      | area=   all    | maxDets= 10 ]    = 0.691
09 Average Recall     (AR) @[ IoU=0.50:0.95      | area=   all    | maxDets=100 ]    = 0.746
10 Average Recall     (AR) @[ IoU=0.50:0.95      | area= small    | maxDets=100 ]    = 0.083
11 Average Recall     (AR) @[ IoU=0.50:0.95      | area=medium    | maxDets=100 ]    = 0.701
12 Average Recall     (AR) @[ IoU=0.50:0.95      | area= large    | maxDets=100 ]    = 0.813
```

Abbildung 6.12: Ergebnis der Evaluierung zur Objekterkennung auf dem Wafer

Damit wird ebenfalls die Precision über die IoU von 0.75 zur Bewertung des Bildverarbeitungssystems verwendet. Weitergehend wird diese über alle Objekte mit der Objektgröße *large* gebildet und muss einen Mindestwert von 0.7 aufweisen. Die Bildung über die Objektzahl im Bild mit maximal 100 Detektionen und die Bildung über die Objektgröße

large werden zur Ermittlung des Recalls verwendet. Auch hier muss der Wert mindestens 0.7 oder besser betragen, um die Genauigkeit des Bildverarbeitungssystems von 70 Prozent zu erreichen. Die in grün markierten Werte in Abbildung 6.12 liegen alle im Genauigkeitsbereich von 70 Prozent oder höher, wodurch beginnend festgehalten werden kann, dass auch das Training des Anwendungsfalls *Wafer* erfolgreich durchgeführt und evaluiert wurde.

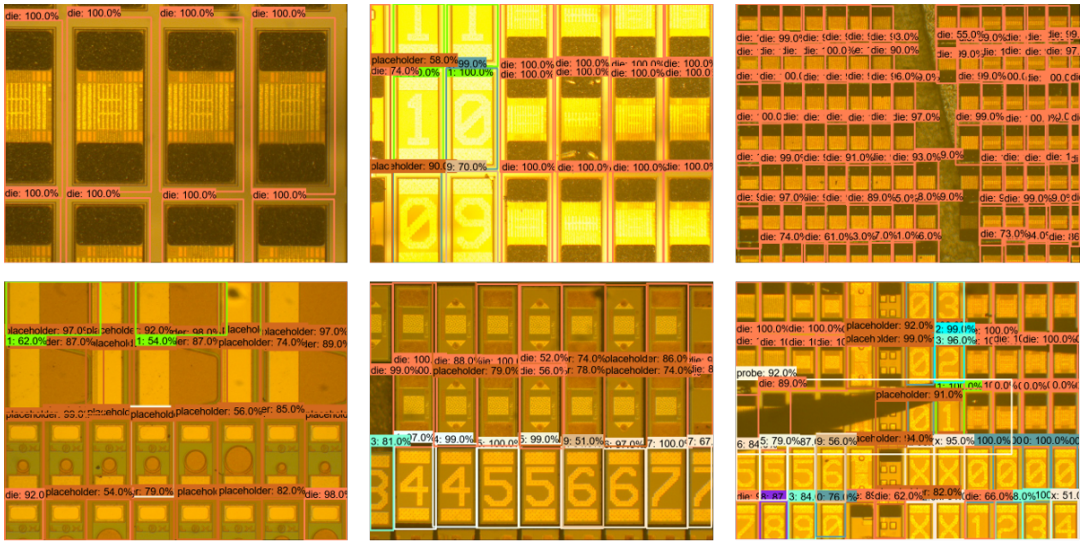

Abbildung 6.13: Anwendung der Objekterkennung für DUT auf dem Wafer auf Bilder im Evaluierungsdatensatz Teil 1

In Abbildung 6.13 ist das Labeling des trainierten neuronalen Netzes für den Anwendungsfall *Wafer* auf einige Evaluierungsdaten angewendet. Hier wird die Klassifikation *die* in braun umrahmt. Oben links im Bild ist zu erkennen, dass die Klassifikation *die* sowohl bei den im Bild erkennbaren DUT durchgeführt wird, als auch bei den DUT im Randbereich, welche abgeschnitten im Bild zu finden sind. Jede dieser Klassifikationen wird mit einer Wahrscheinlichkeit von 100 Prozent bewertet.

Im Evaluierungsbild unten links ist zu erkennen, dass die Objektfindung und -trennung bei MPW zum Teil funktioniert. Bei den vertikal angeordneten kleineren DUT werden zwar viele Fehlklassifikation durchgeführt, jedoch sind die Objekte sehr präzise gerahmt. Diese Erkennung ist daher als Grundlage für eine Korrekturfahrt geeignet. Die Objektrahmen zu den größeren, horizontal angeordneten DUT im oberen Bereich des MPW sind jedoch nicht ausreichend präzise. Einige Objektrahmen umfassen beispielsweise nur einen

Teil des Objektes. Für die Anwendung wird daher gefordert, die DUT im Aufbau horizontal anzuordnen. Speziell für die Bearbeitung dieses MPW, müsste der Wafer um 90 Grad gedreht werden, damit die aktuell horizontalen DUT vertikal angeordnet sind. Außerdem muss die Beleuchtung angepasst werden. Die Umsetzbarkeit der Korrekturfahrt mit der Objekterkennung auf diesem Wafer ist in Abbildung E.2 zu sehen.

Die Genauigkeit des Labelings von Zahlen auf dem Wafer ist in den Evaluierungsbildern hoch. Der Rahmen um diese Objekte ist sehr präzise. Die Klassifikation ist bei einigen Objekten falsch. Hierzu wird ein Folgealgorithmus implementiert werden, der eine Plausibilitätsprüfung jeder Klassifikation durchführt. Für diesen Folgealgorithmus sei auf den Quelltext auf der DVD zu dieser Masterthesis im Modul *object_ detection_ wafer.py* hingewiesen. So ist es nach dem Koordinatensystem der MPW sehr unwahrscheinlich, dass zwischen einer Klassifikation *fünf* und *sechs* eine Klassifikation *neun* liegt, wie es im Bild in der Mitte unten der Fall ist.

Die Ergebnisse in Abbildung 6.13 für den Anwendungsfall *Wafer* liefern exakte Rahmen um die Objekte im Bild, welche für die Korrekturfahrt mit dem Chuck notwendig sind. Lediglich bei MPW muss der*die Nutzer*in die Objektanordnung im Aufbau ändern, sodass die Objekte präzise erkannt werden. Da die Auswahl der zu messenden DUT weiterhin von dem*der Nutzer*in vorgegeben wird, ist die Klassifikation der Objekte nicht notwendig. Lediglich die Klassifikation *probe* soll aus den Daten separiert werden können. Diese ist eindeutig separierbar in dem Bild unten rechts zu finden.

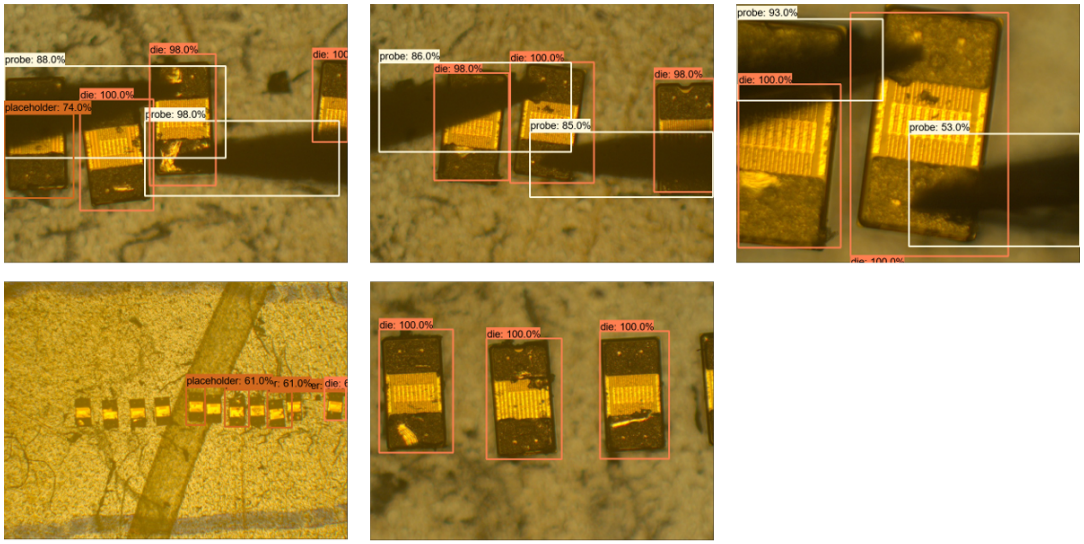

Abbildung 6.14: Anwendung der Objekterkennung für DUT auf dem Wafer auf Bilder im Evaluierungsdatensatz Teil 2

In diesem zweiten Teil sind weitere Beispiele aus den Evaluierungsdaten zum Anwendungsfall *Wafer* zu finden. Hier wird das neuronale Netz auf vereinzelte DUT angewendet, um das trainierte Netz auf die Robustheit zu testen. In den oberen Bildern ist zu erkennen, dass der Rahmen und die Klassifikation *probe* und *die* richtig sind. Auch überlappende Objekte wie im Bild in der Mitte oben sind sehr präzise gerahmt. Im Bild oben links sind die Rahmen bei den überlappten Objekten nicht ausreichend präzise, jedoch sind dort die Messspitzen falsch angeordnet. Im realen Aufbau würde der*die Nutzer*in die Messspitzen aus den jeweiligen Ecken des Bildes zum Kontaktierungspunkt führen, wodurch deutlich geringere Objektüberschneidungen im Bild vorkommen würden. Im Bild unten links ist wieder die Objektgröße *small* zu erkennen, wodurch wie erwartet falsche Klassifikation durchgeführt werden. Außerdem werden einige Objekte aufgrund der geringen Größe nicht erkannt.

Im Vergleich von Abbildung 6.11 und Abbildung 6.14 zeigt das Bild in der Mitte oben, dass die Rahmen mit dem trainierten neuronalen Netz auf 15 verschiedene Klassifikationen präziser bei überlappenden Objekten ist. Außerdem wird in dem Bild oben links von diesem neuronalen Netz ein Objekt mehr erkannt als von dem neuronalen Netz für die Anwendung *vereinzelte DUT*. Schlussendlich kann das neuronale Netz für die Objekterkennung auf Wafern für beide Anwendungsfälle verwendet werden.

6.4 Einbinden des Bildverarbeitungssystems in das Gesamtprogramm

In der Gesamtanwendung soll das trainierte Netz dafür sorgen, dass die Messspitzen sowohl bei *vereinzelten DUT* als auch bei *Wafern* mittig auf dem DUT positioniert werden. Während es für den*die Messingenieur*in beim händischen Kontaktieren immer eine intuitive Entscheidung war, wo sich die Mitte des DUT befindet, um die Messspitzen auf den Pads zu kontaktieren, trifft diese Entscheidung in der Automatisierungslösung das Bildverarbeitungssystem. Dabei bedeutet mittig nicht, dass die Messspitzen exakt auf der Mitte des Pads liegen müssen, sondern dieses lediglich sicher treffen sollen. Da der*die Nutzer*in die Erstkontaktierung händisch durchführt, ist der Abstand der Messspitzen so vorbereitet, dass diese auf den Pads des DUT landen, wenn die richtigen Positionswerte zu dem DUT vorliegen und die Ausrichtung des Wafers exakt ist.

6.4.1 Wafer

In der Anwendung *Wafer* soll nach jeder Positionsanfahrt kontrolliert werden, ob sich das DUT unter den Messspitzen befindet. Hierzu wird nach Beendigung der Fahrt des Chucks zum DUT das aktuelle Bild aus der Kamera gelesen. Über das neuronale Netz wird die Objekterkennung auf das ausgelesene Bild angewendet.

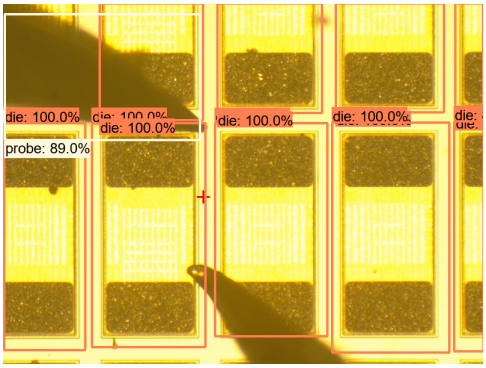

Abbildung 6.15: Positionsanfahrt des Chucks mit Labeling durch trainiertes neuronales Netz

Das rote Kreuz in Abbildung 6.15 wurde von dem*der Nutzer*in in der Konfigurationsphase der Automatisierung im Bild festgelegt. Dabei setzt der*die Nutzer*in das Kreuz

auf den Mittelpunkt zwischen den Messspitzen. Das Kreuz wird als Referenz zur Ermittlung einer erfolgreichen Kontaktierung verwendet. Wie in Abbildung 6.15 zu erkennen ist, liegt der Mittelpunkt der Messspitzen nicht mittig auf dem DUT. Über einen Algorithmus wird entschieden, zu welchem DUT im Bild die Korrekturfahrt durchgeführt werden muss. Dabei werden zu den einzelnen Objektrahmen die Mittelpunkte ermittelt, wie es in der folgenden Abbildung zu erkennen ist. Für den Algorithmus sei auf den Quelltext im Modul *object_ detection_ wafer.py* auf der DVD zu dieser Masterthesis hingewiesen.

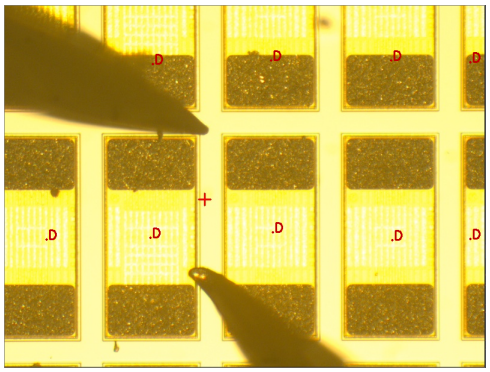

Abbildung 6.16: Ermitteln der Mittelpunkte der Objektrahmen

Zu jedem dieser Punkte wird die Strecke zum Kontaktierungsmittelpunkt der Messspitzen ermittelt. Die kürzeste Entfernung zwischen Kontaktierungsmittelpunkt und Objektmittelpunkt bildet die Korrekturfahrt. Dabei werden nochmal die einzelnen Werte der Korrektur in X- und in Y-Richtung ermittelt. Abschließend werden die in Pixel ermittelten Werte über den Skalierungsfaktor in das Koordinatensystem des Chucks umgerechnet.

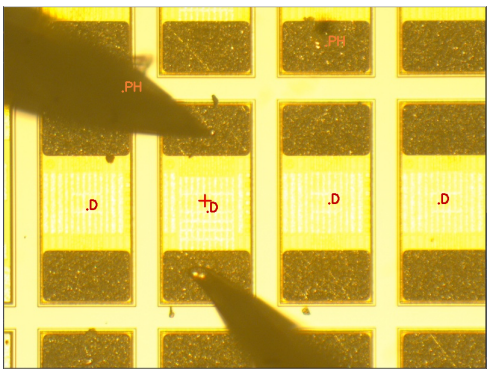

Abbildung 6.17: Beendigung der Korrekturfahrt nach Labeling des trainierten neuronalen Netzes

Nach der Korrekturfahrt ist weiterhin eine Differenz zwischen dem Mittelpunkt des Objektrahmens und dem Kreuz für die Kontaktierung zu sehen. Diese Differenz wurde bereits bei der Erstkontaktierung ermittelt, da der gesetzte Mittelpunkt durch den*die Nutzer*in und der ermittelten Mitte durch das neuronale Netz nie positions-gleich sind. Zudem ist eine Positionsprognose eines Objektes durch das neuronale Netz nicht zwangsläufig deckungsgleich mit der Positionsprognose des Objektes im nächsten Bild der Kamera, da sich beispielsweise die Lichtverhältnisse verändern. Somit wurde eine Korrekturfahrt erfolgreich durchgeführt und beendet. Die Messspitzen befinden sich auf den Pads des DUT wie gefordert.

6.4.2 Vereinzelte DUT

Bei den vereinzelten DUT wird über das Bildverarbeitungssystem die Position des nächsten DUT ermittelt. Hierfür wird nach durchgeführter Messung auf das aktuelle Bild der Kamera die Objekterkennung angewendet.

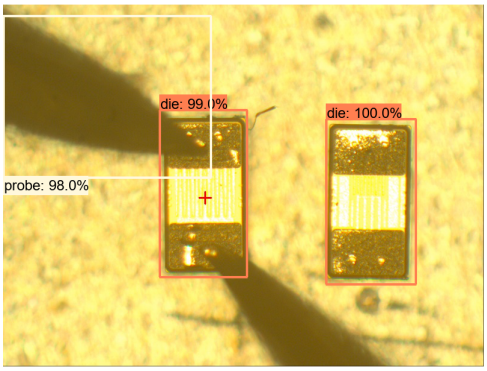

Abbildung 6.18: Objektfindung bei vereinzelten DUT

Über das gesetzte Kreuz als Referenz, die durchschnittliche Objektgröße und den Abstand der einzelnen Objekte zum Kontaktierungspunkt kann das rechts nächstliegende DUT zum Kontaktierungsmittelpunkt ermittelt werden.

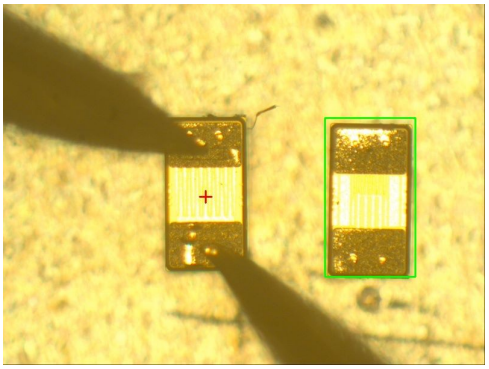

Abbildung 6.19: Nächstes zu kontaktierendes DUT

In grün durch die Berechnung eingerahmt, ist das rechts nächste zu kontaktierende DUT zu erkennen. Über die bekannte Skalierung kann der Abstand zur aktuellen Position im Koordinatensystem des Chucks kalkuliert und über eine relative Fahrt kontaktiert werden.

Mit den trainierten neuronalen Netzen lassen sich zusammenfassend die DUT im Bild finden, wodurch sich beim Wafer eine Korrekturfahrt realisieren lässt. Bei vereinzelten DUT lässt sich der nächste Kontaktierungspunkt ermitteln. Die Messspitzenkontaktierung auf einem DUT wird somit über das Bildverarbeitungssystem mit trainiertem neuronalen Netz sichergestellt.

7 Testen

Um die gesamte Software zu testen, wurde aus der Definition des Pflichtenheftes der Testplan erstellt, welcher in Abschnitt 7.1 erläutert wird. Hierbei existiert zu jedem Punkt im Pflichtenheft mindestens eine Testaufgabe, um die Funktionalität des Moduls stichprobenartig zu verifizieren. Erste bearbeitete Aufträge mit der Automatisierung sollen die Stabilität der Software unter Beweis stellen. Hierzu findet sich eine Beschreibung in Abschnitt 7.2. Außerdem werden zwei Messingenieure*innen die Software über einen Monat im täglichen Laborbetrieb testen und somit einem Langzeittest unterziehen. Informationen hierzu sind in Abschnitt 7.3 zu finden.

7.1 Testplan

Im Testplan sind zwölf Spalten zu finden. In der Spalte *Unteraufgaben* werden die einzelnen Tests den Unteraufgaben aus dem Aufgabenplan zugeordnet. Über die Testnummer (T-XX) bekommt jeder Test eine Identifikationsnummer. In der Spalte *Unteraufgabe des Prototypen aus Aufgabenplan* wird textuell festhalten, für welche spezifische Aufgabe dieser Test existiert. Mit *Verknüpfung zur technischen Anforderung* wird die Verknüpfung zum Pflichtenheft festgehalten. In *Funktionsbeschreibung* wird detailliert beschrieben, welche Funktionalität der Test beweisen muss. Des Weiteren werden Plattformen festgehalten, auf denen der Test durchgeführt wird und auf denen das Testergebnis zu finden ist. Abschließend wird die Testdurchführung textuell beschrieben. In der vorletzten Spalte wird das Testergebnis über ein „JA/NEIN“ festgehalten, wobei Kommentare in der letzten Spalte abgelegt werden können.

Zu dieser Masterthesis wurden alle definierten Tests mit „JA“ bewertet, sodass das geforderte Pflichtenheft funktional umgesetzt wurde.

7.2 Stabilitätstest

Für den Stabilitätstest der Software wurden mehrere reale Messaufträge mit der Automatisierungssoftware bearbeitet. Hierbei sollte beispielsweise ein Wafer mit einem System mit den Abmaßen $0.33\mu mX0.63\mu m$ vollständig gemessen werden. Auf diesem Wafer sind circa 150.000 Systeme realisiert. Da das Starten, Durchführen und Ablegen der geforderten Messung mit dem SPA 30 Sekunden pro Messdurchlauf benötigt, beläuft sich die theoretische Testzeit ohne Pause auf 30 Tage.

$$Messzeit_{theoretisch} = \frac{150.000 \cdot (15sec + 2.5sec)}{60\frac{sec}{min} * 60\frac{min}{h} * 24\frac{h}{Tag}} = 30\,Tage \tag{7.1}$$

Mit der Software wurden schlussendlich aus Zeitgründen 100.000 DUT gemessen, wobei die Software über 20 Tage getestet wurde. Während des Stabilitätstest sind vor allem Komplikationen im Kommunikationsprotokoll mit dem Chuck und dem SPA aufgefallen, die langfristig behoben werden konnten. Auch für die Messdatenverarbeitung, welche die Schnittstelle zwischen der Automatisierungslösung und dem Ziel der Messung ist, konnten erste Rahmenbedingungen durch den Stabilitätstest festgelegt werden.

7.3 Langzeittest

Beim Langzeittest verwenden zwei Messingenieur*innen die entwickelte Software mit einer angefertigten Bedienungsanleitung. Hierbei wird vor allem die Handhabung der Software und die Richtigkeit der Dokumentationen und Arbeitsanweisungen überprüft. Dadurch, dass neue Nutzer*innen die entwickelte Software verwenden, wird diese erneut auf die Anwendbarkeit getestet. Somit können vor allem Bedienungsroutinen des Entwicklers gegengeprüft werden. Beim Langzeittest wird außerdem die Einstellung der Hardware trainiert. Hierbei lernt der*die Nutzer*in beispielsweise, wie die optische Einheit eingestellt werden muss, sodass das Bildverarbeitungssystem die DUT auf dem Wafer erkennt.

Für detailliertere Informationen zur Testphase der Software sei auf den Testplan hingewiesen, der auf der beigefügten DVD zu dieser Masterthesis unter *Masterthesis:/97_ Dokumentation/Lastenheft_ Plichtenheft_ Testplan.xlsx* zu finden ist.

8 Reflexion

Nachdem die Testphase erfolgreich durchgeführt wurde, kann im Folgenden die Reflexion des Gesamtprojektes betrachtet werden. Hierbei wird zunächst die Objekterkennung in der Bildverarbeitung reflektiert und abschließend die Gesamtanwendung.

Insgesamt ist das durchgeführte Projekt durch die Anwendbarkeit der Software im täglichen Laborbetrieb ein Erfolg. Durch die Erstellung des Pflichtenheftes wurde der Rahmen der Aufgabe sinnvoll festgelegt. Mit der Verwendung des Aufgabenplans konnten die einzelnen Unteraufgaben aus dem Pflichtenheft frühzeitig benannt und konzeptioniert werden. Außerdem konnte durch die Definition des Testplans im Voraus festgelegt werden, wie die erfolgreiche Umsetzung einer Unteraufgabe bemessen werden kann. Durch die Vorausplanungen konnten somit alle Schritte bis zur Fertigstellung des Projektes betrachtet werden, wodurch in der Projektumsetzung keine signifikanten Änderungen im Projektplan vorgenommen werden mussten.

8.1 Objekterkennung als Bildverarbeitungssystem

In Bezug auf das Training der neuronalen Netze ist festzuhalten, dass die Anforderung einer Genauigkeit von 70 Prozent erreicht wurde. Des Weiteren konnten zwei unterschiedliche Trainingsdurchläufe durchgeführt werden, wobei das jeweilige Netz auf den gleichen Trainingsdatensatz optimiert wurde, diese jedoch unterschiedliche Klassifikationsaufgaben zu dem Datensatz besitzen. So findet das neuronale Netz zur Anwendung *vereinzelte DUT* die Klassifikationen *die* und *probe* in dem Bild. Das Netz zur Anwendung *Wafer* soll dagegen vierzehn unterschiedliche Klassifikationen finden. Hierzu zählen ebenfalls die Klassifikationen *die* und *probe*, welche in dieser Anwendung um die Klassifikationen von den Zahlen null bis neun, dem Buchstaben x und *placeholder* erweitert werden. Schlussendlich konnten in dieser Masterthesis mit einem Datensatz zwei unterschiedliche Aufgaben gelöst werden, wobei die eine Aufgabe lediglich eine Teilmenge der Anderen ist.

Die Wahl der Objekterkennung als Bildverarbeitungssystem wurde bereits in der Planungsphase dieser Masterthesis getroffen. Um bei den vereinzelten DUT die Positionierung noch genauer auszulegen, könnte eine Segmentierung als Bildverarbeitungssystem verwendet werden, da so auch die Ausrichtung des DUT festgestellt werden kann. Solange der*die Nutzer*in jedoch die DUT in einer ausreichend genauen Ausrichtung aufreiht, wobei der*die Nutzer*in keine Hilfsmittel benötigt, setzt die Kontaktierung über die Objekterkennung das Pflichtenheft in allen Punkten um. Neben der Umsetzung der Objekterkennung können noch weitere Parameter im Bildverarbeitungssystem optimiert werden, welche im Kapitel 9 vorgestellt werden. Allgemein ist zu erwähnen, dass in diesem Projekt ein Ansatz, bis zur erfolgreichen Umsetzung der Anforderungen, verfolgt wurde. Strategisch wurde dabei der Trainingsdatensatz so lange erweitert, bis das trainierte Netz die geforderte Genauigkeit von 70 Prozent erreicht hat. Schlussendlich hat die Erweiterung der Trainingsdaten zu dem Erfolg der Genauigkeit beigetragen und konnte als sinnvolle Strategie während der einzelnen Trainingsintervalle festgestellt werden.

8.2 Gesamtanwendung

Die Realisierung der Messgerätesteuerung des SPAs bietet dem*der Nutzer*in viele verschiedene Möglichkeiten zur Steuerung und Messdatenablage. Die Kommunikation ist durch den Stabilitätstest ausreichend getestet und somit stabil. Ebenso gilt dies für das Modul *chuck_ hardware.py*, bei dem Protokollaussetzer während der Projektierungsphase, eine große Hürde darstellten. Durch die Realisierung des Threadings der einzelnen Kommandos für den Chuck konnte schlussendlich eine Stabilität der Kommunikation umgesetzt werden, die während des Stabilitätstests unter Beweis gestellt wurde. Das Kameramodul konnte ohne Problemstellungen umgesetzt werden und bietet in der Anwendung die Grundlage für das Bildverarbeitungssystem. Durch Implementierung einer Scroll-Leiste kann der*die Nutzer*in die Belichtungszeit der Kamera in der GUI einstellen, wodurch der*die Nutzer*in eine weitere Möglichkeit hat, das Bildverarbeitungssystem zu beeinflussen. Die umgesetzte Lichtsteuerung erfüllt die geforderten Punkte aus dem Pflichtenheft. Zukünftig soll die aktuelle umgesetzte Hardware mit einer schaltbaren Steckdose durch eine Leuchtdiode ersetzt werden, da das aktuelle Leuchtmittel eine deutlich geringere Lebensdauer als eine Leuchtdiode aufweist.

Die entwickelte GUI konnte ebenfalls ohne große Hürden nach dem Projektplan entwickelt werden. Die unterschiedlichen Schnittstellen, wie beispielsweise die Messgeräte-

schnittstelle konnte sich schon in den ersten Tests als sinnvoll herausstellen. Vor allem der Register-Reiter Wafermap bietet dem*der Nutzer*in die nötige Schnittstelle, um die zu messenden DUT für einen Auftrag auszuwählen. Die Schnittstelle zur Kamera ist ebenfalls ein wichtiges Element der GUI, um den Messverlauf zu beobachten. Durch den Langzeittest konnte die Handhabung schlussendlich auf die Nutzer*innen angepasst werden.

Integriert in die Gesamtanwendung, konnte das umgesetzte Bildverarbeitungssystem die geforderten Aufgaben aus dem Pflichtenheft umsetzen. So wird bei dem Messen der DUT auf dem Wafer mit dem Bildverarbeitungsalgorithmus geprüft, ob sich die Messspitzen mittig über dem DUT befinden. Bei vereinzelten DUT findet der Algorithmus das nächste DUT, welches kontaktiert werden soll. Sollte die Bildverarbeitung das nächste DUT nicht finden, muss der*die Nutzer*in die Position des nächsten DUT manuell bestimmen. In dem Langzeittest konnte jedoch festgestellt werden, dass die manuelle Eingabe der Positionsdaten durch die richtige Einstellung des Objektivs und der Belichtungszeit der Kamera auf ein Minimum reduziert werden kann. Dabei konnte bei vier von fünf Messaufträgen festgestellt werden, dass alle DUT gefunden wurden. Lediglich bei einem DUT in einem der fünf Messaufträge musste der*die Nutzer*in eine manuelle Eingabe tätigen. Das DUT wurde aufgrund von Reflexionen auf der Oberfläche des DUT nicht erkannt. Das Bildverarbeitungssystem bildet die Grundlage zur Kontaktierungskorrektur bei der Anwendung Wafer und eine Messspitzenpositionierung bei vereinzelten Dies, welche händisch aufgereiht auf einer Folie liegen. So wurde in dieser Masterthesis für das Messlabor das erste automatische Kontaktiersystem für vereinzelte, freiliegende DUT entwickelt. Weitere positive Aspekte der Messautomatisierung sind im Folgenden aufgelistet.

1. Es können deutlich größere Messaufträge als mit der händischen Messung abgearbeitet werden.

2. Die Ausnutzung des Messplatzes steigt, da durch größere Messaufträge auch längere Messzeiten entstehen, welche wiederum über Nacht oder am Wochenende durchgeführt werden können.

3. Durch die Nutzung der Nacht- und Wochenendzeit mit der Automatisierung steigt die mögliche Arbeitszeit der Maschine, zu einer vorherigen wöchentlichen Betriebszeit von 40 Stunden, um mehr als 100 Prozent an.

4. Die generierte Messdatenmenge der Arbeitsmaschine steigt somit durch die Automatisierung.

9 Ausblick

Mit dem trainierten neuronalen Netz bieten sich verschiedene Möglichkeiten für weitere Anwendungen. Im Folgenden sind einige Erweiterungen und Neuentwicklungen in Stichpunkten festgehalten.

1. Erweiterung der Automatisierung um automatische Ermittlung des Kontaktierungspunktes.
2. Verbesserung der optischen Erkennung durch ein zweites neuronales Netz zur Segmentierung.
3. Automatisiertes Ausrichten des Wafers ohne die Interaktion mit dem*der Nutzer*in.
4. Optimierung der Hyperparameter und Verändern der Netzarchitektur mit erneutem Training zur Leistungssteigerung der Objekterkennung.
5. Automatisches Generieren einer Wafermap über die optische Kontrolle.

Über die bereits implementierte Objekterkennung der Messspitzen mit der Klassifikation *probe* können beide Positionen der Messspitzen im Bild ermittelt werden. Ein Folgealgorithmus kann über die beiden Objekte ermitteln, wo sich der Kontaktierungsmittelpunkt befindet, wie es im ersten Punkt beschrieben ist. Aktuell muss der*die Nutzer*in das Kreuz zur Markierung des Kontaktierungsmittelpunktes im Bild selbstständig setzen.

Im zweiten Punkt ist erwähnt, dass eine Segmentierung implementiert werden kann. Der Objektrahmen des DUT kann mit dieser Variante rotiert werden, sodass der optimale Kontaktierungspunkt des DUT mit zusätzlicher Betrachtung des Rotationswinkels ermittelt werden kann.

Des Weiteren kann der dritte Punkt implementiert werden. Hierbei kann die Objekterkennung verwendet werden, um das automatische Ausrichten des Wafers zu realisieren. Dabei können die einzelnen Objekte geprüft und die Ausrichtung des Wafers so lange

angepasst werden, bis die einzelnen erkannten Objekte in einer Reihe exakt auf der horizontalen Achse liegen. Die erwähnte Segmentierung aus dem zweiten Punkt kann bei dieser Systemerweiterung ebenfalls eine sinnvolle Bildverarbeitungsgrundlage bieten.

Zur weiteren Stabilisierung des Bildverarbeitungssystems können verschiedene Netzarchitekturen dem entwickelten Training unterzogen werden. Diese erwirken möglicherweise eine höhere Genauigkeit der Objekterkennung. Zusätzlich kann die Optimierung der Hyperparameter zu einer Verbesserung der Genauigkeit beitragen.

Außerdem kann mit dem fünften Punkt ein neues System entwickelt werden, bei dem die Objekterkennung verwendet wird, um automatisch eine Wafermap zu generieren. Hierbei ist beispielsweise das Format *sinf* das Ziel der Software. In dieser *sinf*-Datei befinden sich Informationen zu dem Wafer die in Folgeanwendungen verwendet werden. So könnte die Objekterkennung verwendet werden, um die Positionen aller DUT einer speziellen Größe auf einem MPW in der *sinf*-Datei abzulegen.

Da die entwickelte Software aus dieser Masterthesis für das Messlabor entwickelt wurde, beginnt jetzt die Phase der Anwendung. Der nächste Schritt ist die Implementierung der Software auf einem neuen Rechnersystem, welches mit einer leistungsfähigeren Grafikkarte die Bildwiederholrate des Bildverarbeitungssystems beschleunigen soll. Die Implementierung eines weiteren Messgerätes zur Kapazitätsmessung ist ein weiterer Punkt, der in den kommenden Monaten umgesetzt wird. Für diese Masterthesis bedeutet dies, dass diese die Grundtechnologie für viele weitere Folgeanwendungen darstellt.

Literaturverzeichnis

[1] Basler AG. pylon Open Source Projekte, August 2020. URL https://www.baslerweb.com/de/produkte/software/basler-pylon-camera-software-suite/pylon-open-source-projekte/. [online].

[2] Basler AG. pypylon, 2020. URL https://github.com/basler/pypylon. [online].

[3] Basler AG. a2A1920-160ucBAS, Januar 2021. URL https://docs.baslerweb.com/a2a1920-160ucbas. [online].

[4] Manjo Balakrishnan. *Die Assembly Design Rule for Power Packages and Clip Bounded Packages*. Nexperia, 2020. [firmeninternes Dokument, zu finden auf der DVD zur Masterthesis].

[5] Sara Beery, Guanhang Wu, Vivek Rathod, Ronny Votel, and Jonathan Huang. Context R-CNN: Long Term Temporal Context for Per-Camera Object Detection, April 2020. URL https://arxiv.org/abs/1912.03538. [Paper].

[6] Torsten Bronger and Gregor Thalhammer. PyVISA 1.10.0, August 2019. URL https://pypi.org/project/PyVISA/1.10.0/. [online].

[7] Inc. Cascade Microtech. *REL 6100/4800 Users Guide*. Cascade Microtech, Inc., Mai 2006. [Handbuch].

[8] SCPI Consortium. Standard Commands for Programmable Instruments (SCPI), 1999. URL https://www.ivifoundation.org/docs/SCPI-99.PDF. [online].

[9] Prof. Dr.-Ing. Dipl.-Kfm. Joerg Dahlkemper. Angewandte Industrielle Bildverarbeitung - 9 Numerische Klassifizierung. [Vorlesungsmaterial], März 2019.

[10] Prof. Dr.-Ing. Dipl.-Kfm. Joerg Dahlkemper. Angewandte Industrielle Bildverarbeitung - 10 Einführung in Künstliche Neuronale Netze. [Vorlesungsmaterial], März 2019.

[11] Prof. Dr.-Ing. Dipl.-Kfm. Joerg Dahlkemper. Angewandte Industrielle Bildverarbeitung - 5 Beleuchtung. [Vorlesungsmaterial], März 2019.

[12] Prof. Dr.-Ing. Dipl.-Kfm. Joerg Dahlkemper. Angewandte Industrielle Bildverarbeitung - 8 Merkmalsextraktion. [Vorlesungsmaterial], März 2019.

[13] Prof. Dr.-Ing. Dipl.-Kfm. Joerg Dahlkemper. Angewandte Industrielle Bildverarbeitung - 7 Segmentierung. [Vorlesungsmaterial], März 2019.

[14] Prof. Dr.-Ing. Dipl.-Kfm. Joerg Dahlkemper. Angewandte Industrielle Bildverarbeitung - 6 Bildvorverarbeitung. [Vorlesungsmaterial], März 2019.

[15] Michael Drass, Hagen Berthold, Michael A. Kraus, and Steffen Müller-Braun. Semantic segmentation with deep learning: detection of cracks at the cut edge of glass, September 2020. URL https://doi.org/10.1007/s40940-020-00133-7. [Paper].

[16] P. Fischer. *Technologie - Vom Silizium zum Chip*. Uni Heidelberg, 2016. URL https://sus.ziti.uni-heidelberg.de/Lehre/WS1617_VLSI/VLSI_Fischer_09_Manufacturing.pptx.pdf. [Buch].

[17] Henry Ford. *Mein Leben und Werk : Autobiografie eines modernen Unternehmers*. Amra Verlag, 2014. URL http://d-nb.info/1047914069. [Buch].

[18] IVI Foundation. VPP-4.3: The VISA Library, 2020. URL https://www.ivifoundation.org/specifications/. [online].

[19] Rohith Gandhi. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms, 2018. URL https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e. [Artikel].

[20] Olli-Pekka Heinisuo. opencv-python 4.4.0.46, November 2020. URL https://pypi.org/project/opencv-python/4.4.0.46/. [online].

[21] Ulrich Hilleringmann. *Silizium-Halbleitertechnologie - Grundlagen mikroelektronischer Integrationstechnik*. Springer Vieweg, September 2019. URL https://books.google.de/books?id=xcx8BAAAQBAJ&lpg=

PR5&ots=gbj3RDH5Rt&dq=technologie%20-%20vom%20silizium%20zum%20chip&lr&hl=de&pg=PR5#v=onepage&q=technologie%20-%20vom%20silizium%20zum%20chip&f=false. 6., korrigierte und verbesserte Auflage [Buch].

[22] Google Inc. tensorflow 2.0.0, September 2019. URL https://pypi.org/project/tensorflow/2.0.0/. [online].

[23] Kristian Kersting, Christoph Lampert, and Constantin Rothkopf. *Wie Maschinen lernen - Künstliche Intelligenz verständlich erklärt*. Springer Fachmedien Wiesbaden GmbH, ein Teil von Springer Nature 2019, 2019. URL https://doi.org/10.1007/978-3-658-26763-6. [Buch].

[24] Chris Liechti. pyserial 3.4, Juli 2017. URL https://pypi.org/project/pyserial/3.4/. [online].

[25] Riverbank Computing Limited. PyQt5 5.14.2, April 2020. URL https://pypi.org/project/PyQt5/5.14.2/. [online].

[26] Chris Niklas Lucka. *Erstellung eines Steuerprogramms für eine Keyless Entry System Demo auf Basis eines 32-bit Mikrocontrollers*. Hochschule für Angewandte Wissenschaften Hamburg, 2019. URL http://edoc.sub.uni-hamburg.de/haw/volltexte/2019/4713/. [Bachelorthesis].

[27] Josef Lutz. *Halbleiter-Leistungsbauelemente*. Springer, Berlin, Heidelberg, 2006. URL https://doi.org/10.1007/3-540-34207-9. [Buch].

[28] Cascade Microtech. *Cascade Microtech Alessi AUCS Commands User Guide*, 1995. [Handbuch].

[29] Alfred Nischwitz. *Bildverarbeitung Band II des Standardwerks Computergrafik und Bildverarbeitung*. Springer eBook Collection. Springer Vieweg, 4. auflage edition, 2020. URL https://doi.org/10.1007/978-3-658-28705-4. [Buch].

[30] Inc. Phaseit. PyPDF2 1.26.0, März 2016. URL https://pypi.org/project/PyPDF2/. [online].

[31] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2015. URL https://arxiv.org/abs/1506.01497. [Paper].

[32] Prof. Dr. rer. nat. Jörg Frochte. *Maschinelles Lernen - Grundlagen und Algorithmen in Python*. Carl Hanser Verlag München, 2019. URL http://www.hanser-fachbuch.de. [Buch].

[33] Gilbert Tanner. How to train a custom object detection model with the Tensorflow Object Detection API, 2021. URL https://github.com/TannerGilbert/Tensorflow-Object-Detection-API-Train-Model. [online].

[34] Keysight Technologies. *Keysight EasyEXPERT Software - User's Guide Volume 2*, 2020. URL https://www.keysight.com/de/pd-582565-pn-B1500A/semiconductor-device-analyzer?pm=PL&nid=-33019.536905585&cc=DE&lc=ger. [Handbuch].

[35] TensorFlow. TensorFlow Object Detection API, 2020. URL https://github.com/tensorflow/models/tree/master/research/object_detection. [online].

[36] Tzutalin. LabelImg, 2015. URL https://github.com/tzutalin/labelImg. [online].

A Dokumentation - Lastenheft

In dem Lastenheft sind die Anforderungen an die Automatisierung des Darkboxprobers festgehalten. Hierbei hat der Kunde, in diesem Fall das Protection & Filtering Labor von Nexperia Germany GmbH, die aufgelisteten Anforderungen an die Automatisierung festgelegt.

Anforderungs - nummer	Wichtigkeit	Anforderungsbeschreibung
	Ziel	Waverprobeautomatisierung
A-1	muss	Es muss eine Steuerung zur automatisierten Kontaktierung von DUTs mit Messspitzen realisiert werden.
A-2	muss	Die optische Erkennung muss eine Genauigkeit von mindestens 70% aufweisen.
A-3	muss	Es müssen sowohl DUTs auf dem Wafer, als auch gesägte, freiliegende DUTs automatisiert kontaktiert werden können.
A-4	muss	Alle Messskripte des SPAs müssen automatisch am kontaktierten DUT durchgeführt werden können.
A-5	muss	Die Software muss eine Möglichkeit zur Auswahl der zu messenden DUTs bieten
A-6	muss	Die Software muss die Möglichkeit zur Auswahl der durchzuführenden Messung auf dem B1500A bieten.

Abbildung A.1: Lastenheft der Automatisierung des Darkboxprobers

B Dokumentation - Pflichtenheft

Aus den definierten Anforderungen im Lastenheft wurden die technischen Anforderungen im Pflichtenheft abgeleitet.

Technische Anforderungs-nummer	Anforderungs-nummer aus Lastenheft	Wichtigkeit	Funktionsspezifische Anforderungsbeschreibung
		Ziel	Waferprobeautomatisierung
TA-1	A-5, A-6, A-1, A-5, A-5	muss	Über eine GUI muss die Mensch-Maschine-Interaktion realisiert werden. Hierbei definieren die folgenden fünf Andorderungen die GUI
TA-1-1	A-5	muss	In der GUI muss der*die Nutzer*in die zu messenden DUTs auswählen können.
TA-1-2	A-6	muss	In der GUI muss der*die Nutzer*in alle Messkripte des SPAs auswählen können.
TA-1-3	A-1	muss	In der GUI muss der*die Nutzer*in die automatisierte Kontaktierung der DUTs starten können, wobei die Erstkontaktierung von dem*der Nutzer*in manuell durchgeführt werden muss.
TA-1-4	A-5	muss	In der GUI muss dem*der Nutzer*in eine Eingabemöglichkeit für Wafermaps geboten werden, wobei hierfür eine Wafermap im DPL-Format und eine PDF mit Informationen zum Reticle vorliegen müssen.
TA-1-5	A-5	muss	In der GUI muss dem*der Nutzer*in eine alternative Eingabemöglichkeit der Informationen aus der DPL-Wafermap und der PDF-Datei geboten werden.
TA-2	A-4,A-6	muss	Die Software muss alle Messskripte aus dem SPA auslesen können.
TA-3	A-1	muss	Der*die Nutzer*in muss vor dem Start der Automatisierung den aufgelegten Wafer händisch arritieren, sodass dieser parallel zum Koordinatensystem des Chucks im Aufbau integriert ist. Hierbei ist unter parallel die Parallelität der X-Achse des Chucks und die in X-Richtung verlaufenden Sägebahnen auf dem Wafer zu verstehen.
TA-4	A-1, A-2, A-3	muss	Ein Bildverarbeitungsalgorithmus muss die DUTs voneinander separieren können und die Koordinaten im Kamerasystem zu allen erkannten Dies liefern. Dieser Algorithmus muss über ein neuronales Netz realisiert werden und eine Genauigkeit von 70% aufweisen.
TA-5	A-1, A-3	muss	Die Chucksteuerung muss in X-, Y- und Z-Richtung realisiert werden, wobei keine direkte Geschwindigkeitsvorgabe gefordert ist.
TA-6	A-1, A-2, A-3	muss	Die Lichtquelle zur Beleuchtung des Sichtbereiches der Kamera auf dem aufgelegten Wafer des Darkboxprobers muss steuerbar sein.
TA-7	A-1, A-3	muss	Ein Algorithmus muss dafür sorgen, dass die Messspitzen auf dem nächsten DUT positioniert werden. Hierbei darf der Chuck zuerst nur in negative Z-Richung, danach in X- und Y-Richtung verfahren werden. Diese Funktion dient zum Kratzschutz des Wafers, damit dieser bei einer Bewegung in X- und Y-Richtung keine Schäden durch die kontaktierten Messpitzen erfährt.

Abbildung B.1: Pflichtenheft der Automatisierung des Darkboxprobers

C Dokumentation - Aufgabenplan

Der Aufgabenplan wurde aus den Anforderungen aus dem Pflichtenheft erstellt. In diesem ist durch die farbliche Kennzeichnung erkennbar, dass alle Aufgaben durch die Forderungen im Pflichtenheft abgearbeitet wurden. Zu den einzelnen Unteraufgaben in den Teilaufgaben werden Testfälle festgelegt, die im Testplan auf der DVD zur Masterthesis zu finden sind.

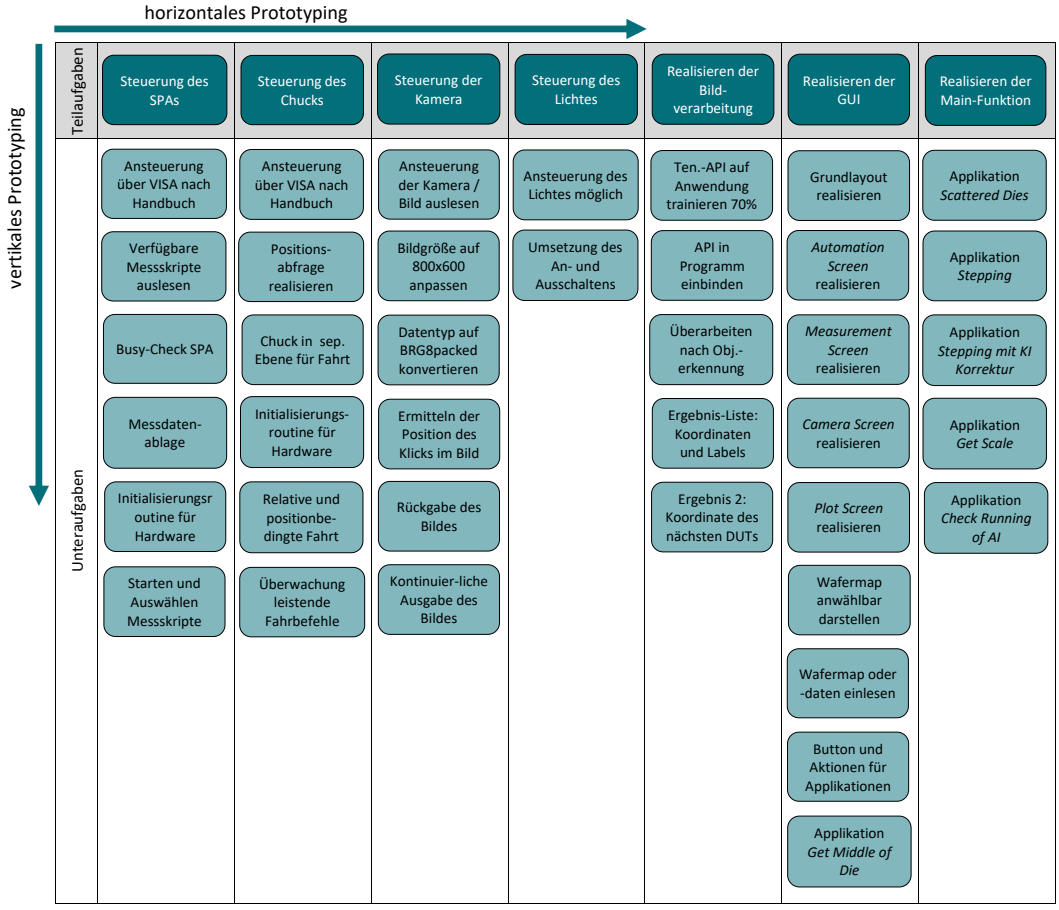

Abbildung C.1: Aufgabenplan der Automatisierung des Darkboxprobers

D Projektpfad Erläuterung

Im Projektpfad sind die einzelnen Module zu finden. Außerdem ist die Tensorflow Object Detection API im Projektpfad abgelegt. In der folgenden Tabelle werden die einzelnen Ordner und Module benannt.

Tabelle D.1: Ordner im Verzeichnis Software

Ordnername	Inhalt und Funktion
01_camera	Ablageordner für das aktuellste Bild für die Objekterkennung
02_documents	Zusatzinformationen und Arbeitsanweisungen
03_template	Bilder in der GUI und exemplarische Wafermaps.
04_printout	Hier wird das Printout der GUI nach einem Automatisierungsdurchlauf abgesichert
05_setups	Ablageort für GUI-Konfigurationen die über *Save GUI* gesichert wurden
models	Integrationsordner der Tensorflow Object Detection API, aufgrund der großen Datenmenge sind nur die trainierten Modelle hier zu finden.

In den folgenden Tabellen wird die grundlegende Funktion der einzelnen Module der Software beschrieben. Aufgeteilt sind die Module in Steuerungsmodule für die Hardware in Tabelle D.1, Module für die Erstellung der gesamten GUI in Tabelle D.3 und die einzelnen Applikationen wie beispielsweise das Stepping über den Wafer in Tabelle D.4.

Tabelle D.2: Module im Verzeichnis Software zur Steuerung der Hardware

Modulname	Funktion
camera_hardware	Auslesen der Kamera.
camera_light_hardware	Steuern des Lichtes.
chuck_hardware	Auslesen und Steuern des Chucks.
spa_hardware	Steuerung und Auslesen des SPAs.

Tabelle D.3: Module im Verzeichnis Software zur GUI

Modulname	Funktion
automation_screen_ui	Erstellen des *Automation Screen* oben links in der GUI.
basic_structure_ui	Erstellen des Layouts der GUI.
camera_screen_ui	Fenster oben rechts in der GUI zur Abbildung des Kamerabildes in der GUI.
color_and_font	Festlegung von Farben und Konfigurationen von Elementen in der GUI wie beispielsweise ein Button
dut_information_ui	Klasse zum Abspeichern der eingegebenen Daten zum DUT beziehungsweise zum Wafer.
measurement_screen_ui	Fenster unten links in der GUI mit den notwendigen Komponenten zur Auswahl der Messskripte.
menu_bar_ui	Erstellung der Aktionsleiste im oberen Bereich der GUI in der alle Aktionen verwaltet werden.
plot_screen_ui	Fenster unten rechts in der GUI zur Darstellung der Messergebnisse.
printout_screen_ui	Registerkarte drei in der GUI zur Dokumentation des Messablaufes (Zusatzmodul, nicht in der Masterthesis beschrieben).
reset_interface_ui	Aktion zum Löschen aller bereits eingegebenen Daten in der GUI.
settings_ui	Modul zum Abspeichern der Einstellungen der GUI.
status_bar_ui	Erstellen der Statusleiste in der GUI in der zum Beispiel Informationen zum Entwicklungsdatum zu finden sind.
status_screen_ui	Zweite Registerkarte unten links in der GUI in der der Status der abgearbeiteten DUT und Reticles zu finden ist.
wafermap_screen_ui	Registerkarte zwei in der GUI in der die Wafermap automatisch generiert wird und der*die Nutzer*in Reticles und DUT für die Messung auswählen kann.

Tabelle D.4: Module im Verzeichnis Software für die einzelnen Applikationen

Modulname	Funktion
align_hand_application	Applikation zum halbautomatischen Ausrichten des Wafers (Zusatzmodul, nicht in der Masterthesis beschrieben).
csv_header_interface	Extra Fenster zur Eingabe von wichtigen Daten für die Messung. Hier werden Daten in den Messdaten gespeichert (Zusatzmodul, nicht in der Masterthesis beschrieben).
main_darkboxprober_application	Erstellen der Gesamtanwendung und das Aufrufen aller Objekte.
object_detection_sawn_wafer	Schnittstellenmodul zur Tensorflow Object Detection API zur Objekterkennung von gesägten Wafern.
object_detection_wafer	Schnittstellenmodul zur Tensorflow Object Detection API zur Objekterkennung von DUT auf dem Wafer.
scale_application	Applikation zum Ermitteln der Skalierung zwischen Kamera- und Chuckkoordinatensystem.
scattered_application	Applikation zum automatischen Kontaktieren vereinzelter DUT.
slave_interface	Schnittstellenmodul zum Betrieb der Software in einem Slave-Modus (Zusatzmodul, nicht in der Masterthesis beschrieben).
slave_stepping_application	Modul zum Stepping über den Wafer im Slave-Modus (Zusatzmodul, nicht in der Masterthesis beschrieben).
stepping_application	Modul zum Stepping über den Wafer.
view_object_detect_application	Applikation zum Abbilden des Labelings bei Wafern.
view_object_detect_scattered_application	Applikation zum Abbilden des Labelings bei gesägten vereinzelten DUT.
wafermap_dpl	Applikatiuon zum Einlesen von Daten zum Wafer aus einem DPL-Format.
wafermap_pdf	Applikatiuon zum Einlesen von Daten zum Wafer aus einem PDF-Format.

E Einstellung der Hardware für eine lauffähige Objekterkennung

Im Folgenden wird mit einigen Beispielen beschrieben, wie die Hardware eingestellt werden kann, sodass das Labeling der Objekterkennung für die Korrekturfahrt verwendet werden kann. Es wird auf den Anwendungsfall Wafer und auf den Anwendungsfall vereinzelte DUT eingegangen.

E.1 Wafer

In der folgenden Abbildung ist zu erkennen, dass die Hardware für das Stepping bereits optimal eingestellt ist.

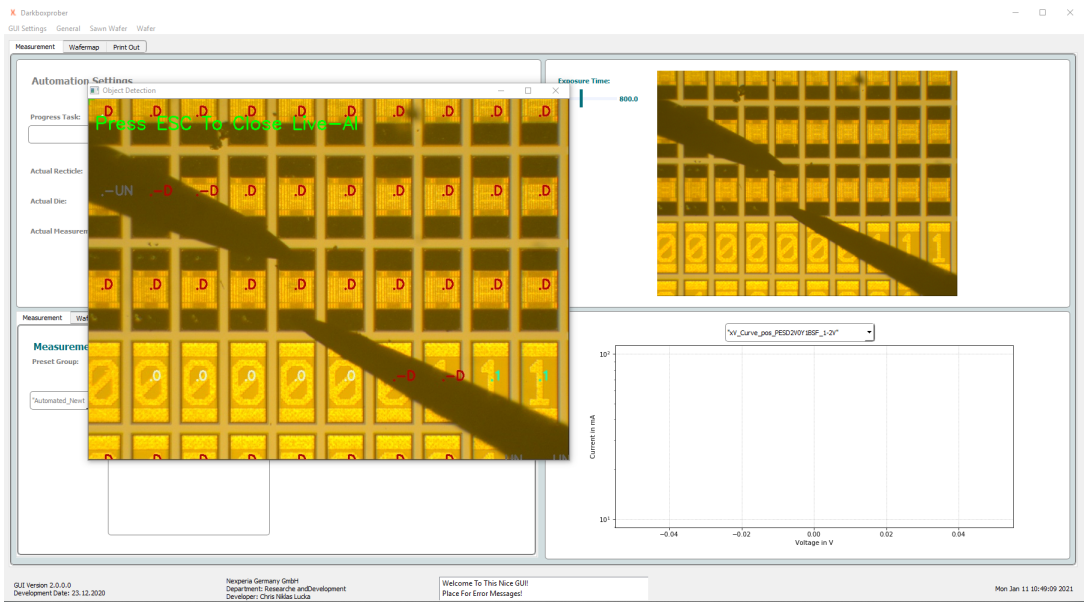

Abbildung E.1: Einstellung der Objekterkennung auf dem Wafer

Im Hintergrund ist die GUI zu sehen, in der das ungelabelte Bild dargestellt wird. Im Vordergrund ist die Applikation *view_ object_ detect_ application* gestartet, in der live das Labeling verfolgt werden kann. Im Falle des Steppings ist nur das Labeling in der kontaktierten Zeile wichtig. Dieses muss stabil in der Nähe des Mittelpunktes des DUT einen Punkt setzen. Die Klassifikation selbst ist für diesem Anwendungsfall nicht entscheidend, solange das DUT nicht als *probe* klassifiziert wird.

Im folgenden Beispiel wird ein Anwendungsfall vorgestellt, in dem das Labeling mit der Startkonfiguration nicht ausreichend eingestellt ist. Dieses Beispiel ist bereits im Kapitel zum Evaluieren der Objekterkennung erwähnt. Dort war der Wafer zu diesem Bild um 90 Grad gedreht, wodurch die Objekterkennung kein erfolgreiches Labeling durchführen konnte.

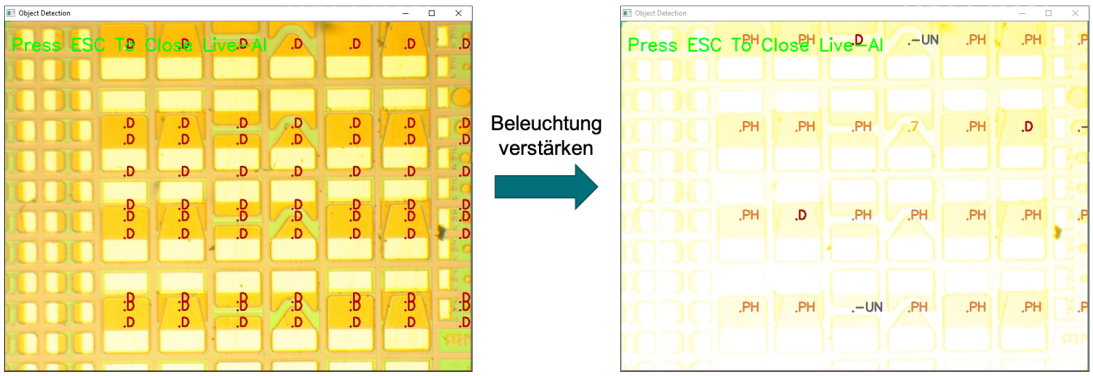

Abbildung E.2: Einstellen der Objekterkennung beim MPW mit unterschiedlichen Systemgrößen durch Erhöhung der Beleuchtungsstärke

In Abbildung E.2 ist zu erkennen, dass durch die Erhöhung der Beleuchtungsstärke viele Informationen im Bild ausgeblendet werden. Durch den Informationsverlust im Bild kann die Objekterkennung Linien im DUT ignorieren, wodurch der Mittelpunkt des DUT im Bild schlussendlich richtig gelabelt wird. Die Rotation des Wafers um 90 Grad hat zur erfolgreichen Umsetzung der Bildverarbeitungsaufgabe beigetragen. Die Klassifikation ist nicht erfolgreich durchgeführt, jedoch ist diese für die Anwendung Wafer nicht notwendig, da lediglich die Objektpositionen verwendet werden. Die Klassifikation *probe* wird nicht durchgeführt, wodurch alle Objektmittelpunkte verwendet werden können.

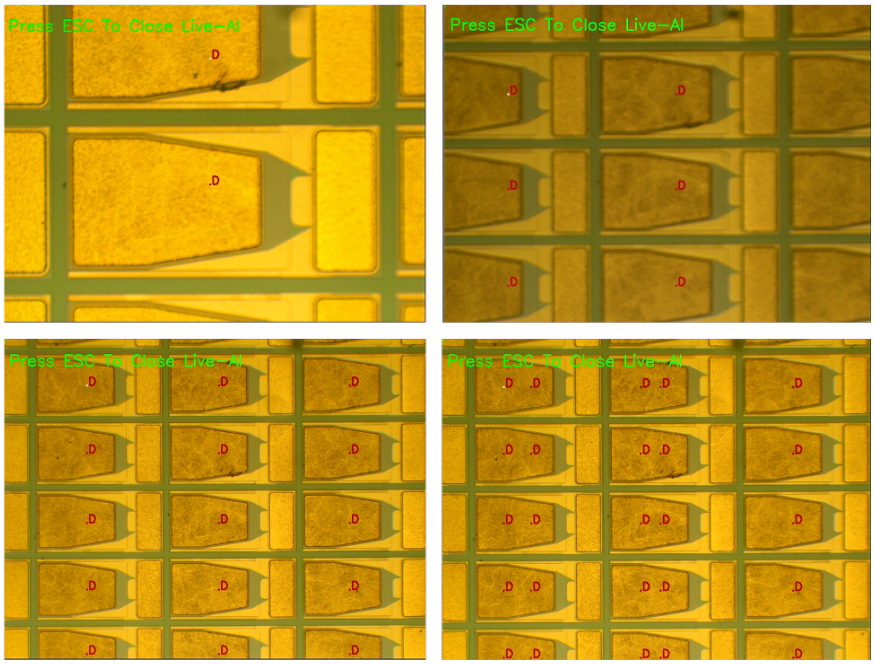

Abbildung E.3: Unterschiedliche Hardwarekonfiguration bei Anwendung der Objekterkennung auf einem Wafer Beispiel 1

In Abbildung E.3 ist exemplarisch zu erkennen, wie unten rechts im Bild das Labeling nicht ausreichend genau durchgeführt wird. Dort werden auf jedem DUT zwei Mittelpunkte gesetzt. Bei einer leichten Vergrößerung des Bildes und Anpassung der Bildschärfe wie es unten links der Fall ist, funktioniert das Labeling ohne einen Aussetzer. Beim Zoom und dem Verwenden von anderen Objektiven, wie es in den zwei oberen Bildern der Fall ist, ist die Objekterkennung ebenfalls stabil und setzt die geforderte Stabilität an das Bildverarbeitungssystem um.

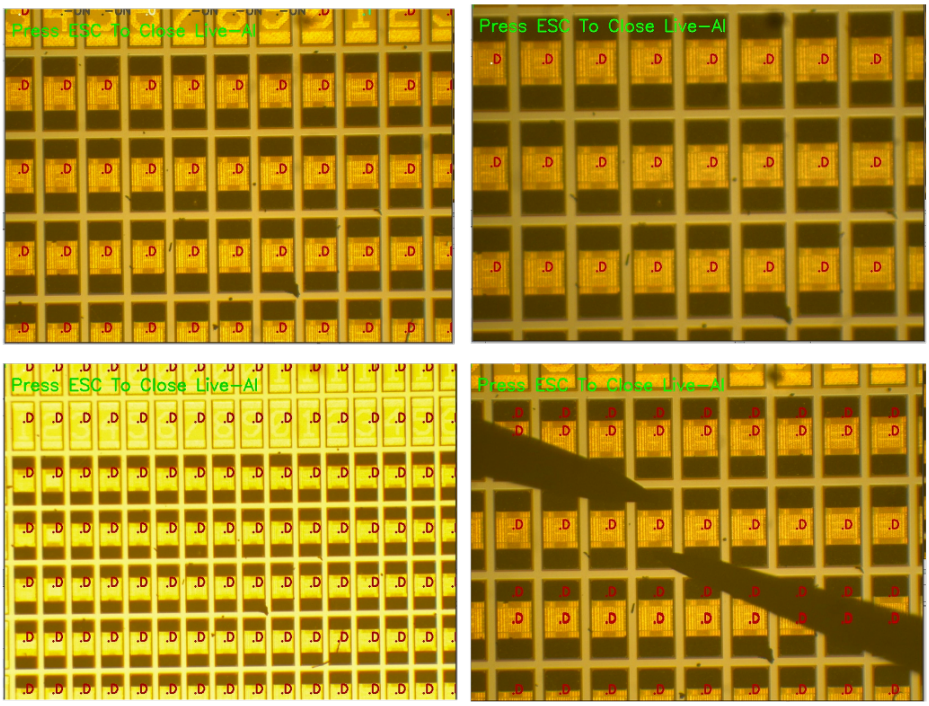

Abbildung E.4: Unterschiedliche Hardwarekonfiguration bei Anwendung der Objekterkennung auf einem Wafer Beispiel 2

In Abbildung E.4 sind wieder unterschiedliche Objektgrößen zu erkennen, die zum Einen durch einen unterschiedlichen Zoom und zum Anderen durch unterschiedliche Objektive verursacht werden. Sobald, wie unten rechts gezeigt, ein Objekt wie eine Probe ins Bild geführt wird, verursacht dies eine Ungenauigkeit in den Überlappungsbereichen der Objekte. Somit ist eine Anforderung an den Aufbau, dass der*die Nutzer*in die Messspitzen so im Bild anordnet, dass diese nicht die Reihe und Spalte des kontaktierten DUT schneiden, da über die Nachbarobjekte der Mittelpunkt des DUT noch einmal überarbeitet und somit stabilisiert wird. Im oberen Bereich vom Bild unten rechts ist zu erkennen, dass mehrere Objektmittelpunkte festgelegt werden. Diese Ungenauigkeit wird durch die überlappende Messspitze über den DUT verursacht. In den anderen drei Bildern ist wieder zu erkennen, dass durch den Zoom die Objekterkennung beeinflusst werden kann. In diesen Beispielen ist die Objekterkennung in jeder Konfiguration sehr stabil.

E.2 Vereinzelte DUT

Bei vereinzelten DUT kann, genau wie bei Wafern, die Wahl des Objektivs und die Einstellung der Beleuchtung in Kombination mit Bildschärfe ausschlaggebend für die erfolgreiche Umsetzung einer Objekterkennungsaufgabe sein.

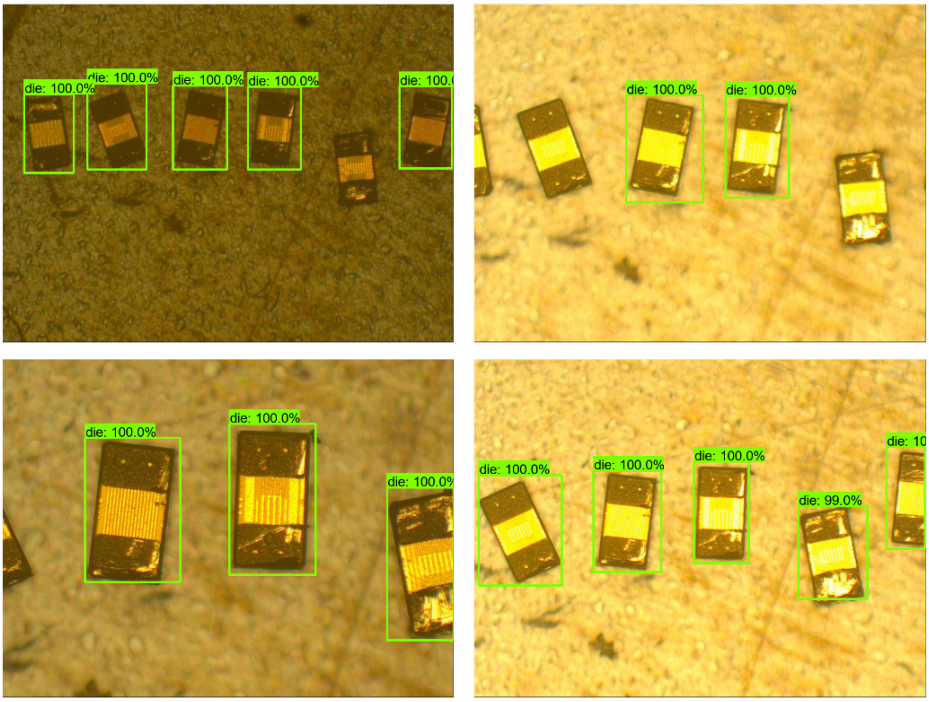

Abbildung E.5: Einstellen der Objekterkennung bei vereinzelten DUT

So ist in Abbildung E.5 im Bild oben links zu erkennen, dass ein DUT nicht gefunden wird. Beim Verwenden eines anderen Objektivs wie unten links zu erkennen ist, in dem die DUT deutlich größer dargestellt werden, wird das DUT doch durch die Objekterkennung gefunden. Ähnlich gestaltet sich die Situation in den beiden rechten Bildern. Auch hier wird im oberen Bild zwei DUT nicht gefunden. Durch Verbesserung der Bildschärfe werden im unteren Bild die beiden DUT im Bild Klassifiziert und grün umrahmt.

Die aufgezeigten Beispiele sollen darstellen, dass die Leistung der Objekterkennung in Abhängigkeit zu den getätigten Hardwareeinstellung steht. Somit ist die Leistung der Objekterkennung abhängig von dem*der Nutzer*in, welche die Objekterkennung im Vorfeld überprüft und entsprechende Konfigurationen der Hardware zur Leistungssteigerung für die jeweilige Aufgabenstellung vornimmt.

F Informationen zur beigefügten DVD

Auf der beigefügten DVD zu der ausgedruckten Version dieser Masterthesis ist die folgende Ordnerstruktur mit den beschriebenen Inhalten zu finden.

- *00_Schriftstück* - Die PDF und Latex-Version dieser Masterthesis.
- *01_Literatur* - Teile der verwendeten Literatur die nicht öffentlich zugänglich sind.
- *97_Dokumentation* - Die Dokumentation zum Projekt.
  - *00_Projektplanung* - Der Projektplan, das Lastenheft, das Pflichtenheft, der Aufgabenplan und der Testplan.
  - *01_Präsentationen* - Präsentationen zum Projekt.
  - *02_Arbeitsanweisung_Bedienung* - Video zum Bedienen der Software.
  - *03_Bilder* - Bilder der Software und der Hardware.
  - *04_Wafermap* - Eine exemplarische Wafermap aus einem Projekt.
  - *05_Python_Package_Dokumentation* - Dokumentation der Python-Package-Installationen der Ausgabe von *conda list*.
- *98_Datensätze* - Die abgespeicherten Trainings-, Test- und Evaluierungsdatensätze.
- *99_Software* - Das lauffähige Python-Programm ohne die installierte Tensorflow Object Detection API im Ordner *models*.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original