

Abschlussprüfung Winter 2022

Fachinformatiker für Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

Entwicklung von PaMesAn

**Implementierung eines neuen Systems zur Erfassung von
Versandverpackungen mit Hilfe von Bild- und Sensordaten zur
Erfüllung der Novelle des Verpackungsgesetzes**

Abgabetermin: Dresden, den 14.11.2022

Prüfungsbewerber

Johannes Leyrer
Krenkelstr. 25
01309 Dresden

Ausbildungsbetrieb

FLYERALARM Industrial Print GmbH
Zschoner Ring 9
01723 Wilsdruff

Dieses Werk einschließlich seiner Teile ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Autors unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen sowie die Einspeicherung und Verarbeitung in elektronischen Systemen.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Projektumfeld	1
1.2	Projektbeschreibung	1
1.3	Projektschnittstellen	2
1.4	Projektabgrenzung	2
2	Projektplanung	2
2.1	Projektphasen	2
2.2	Abweichungen vom Projektantrag	3
2.3	Ressourcenplanung	3
2.4	Entwicklungsprozess	3
3	Analyse	3
3.1	Ist-Analyse	3
3.2	Soll-Analyse	4
3.3	„Make or buy“-Analyse	4
3.4	Analyse der benötigten Hardware	5
3.5	Analyse des Standorts	5
4	Entwurf	6
4.1	Zielplattform	6
4.2	Datenmodell	6
4.3	Architekturdesign	6
4.4	Entwurf des Sensorträgers	7
5	Implementierungsphase	7
5.1	Erstellung der Datenbanktabellen	7
5.2	Erstellen der RabbitMQ Exchanges	7
5.3	Erstellen des Datenspeicherservice	8
5.3.1	Verwendete Pakete	8
5.3.2	Datenbankanbindung	8
5.4	Platzierung der Sensoren und der Webcam	8
5.5	Erstellen der Sensordaten-Abfrage-Anwendung	9
5.5.1	Verwendete Pakete	9
5.5.2	Implementierung der Ausleselogik	9
5.6	Erstellen des Datenverarbeitungsservice	10
5.6.1	Verwendete Pakete	10
5.6.2	Zusammenspiel der einzelnen Komponenten	10
5.6.3	Erkennung der Versandverpackung	11
5.6.4	Auslesen des Versandlabels	12
5.6.5	Berechnung der Paketgröße	12

5.7	Bestückung des Sensorträgers	12
6	Abnahme- und Einführungsphase	13
6.1	Code-Review	13
6.2	Inbetriebnahme und Abnahme	13
7	Dokumentation	13
8	Fazit	14
8.1	Soll-/Ist-Vergleich	14
8.2	Lessons Learned	14
8.3	Ausblick	15
	Eidesstattliche Erklärung	16
	Literatur	IV
	Glossar	V
	Abkürzungsverzeichnis	VI
	Abbildungsverzeichnis	VII
	Tabellenverzeichnis	VIII
	Listings	IX
A	Anhang	i
A.1	Angebot der Elektro Löther GmbH	i
A.2	Auswertung IR-Sensor	i
A.3	Auswertung Ultraschallsensor	ii
A.4	Auswertung LasersSensor	ii
A.5	Tabellenmodell	iii
A.6	Sensorhalterung	iv
A.7	2D-Sensorhalterung	v
A.8	3D-Sensorträger	vi
A.9	Fertiger Sensorträger	vii
A.10	Arduino mit Lasersensoren	viii
A.11	Beispiel Verwendung labelImg	ix
A.12	Beispiel Barcodeerfassung	ix
A.13	Beispiel Versandverpackungserkennung mit YOLOv7	x
A.14	Beispiel Labelerfassung	xi
A.15	GitLab-CI/CD	xii
A.16	SQL-Erstellungsskript für RawData	xiii

B Entwicklerdokumentation PaMesAn	B-1
B.1 Sensorträger und Komponenten	B-1
B.2 Datenspeicherservice	B-4
B.3 Datenausleseservice	B-5
B.4 Datenverarbeitungsservice	B-7

1 Einleitung

Wörter aus dem Glossar, ~~zu~~ sehen in Abschnitt 8.3 auf Seite V, sind zur besseren Übersicht kursiv gesetzt. Abkürzungen sind im Abkürzungsverzeichnis in Abschnitt 8.3 auf Seite VI zu finden. Der Projekttitel „PaMesAn“ ist eine Kurzschreibweise von „Paket-Messungs-Anlage“.

1.1 Projektumfeld

Die FLYERALARM GmbH (FA) ist eines der größten deutschen E-Commerce-Unternehmen und eine der führenden Online-Druckereien Europas im B2B-Bereich. 2002 wurde diese als Ein-Mann-Unternehmen gegründet und beschäftigt mittlerweile mehr als 2000 Mitarbeitenden (MA). Das Produktsortiment umfasst drei Millionen Produkte und wird stetig erweitert. Dazu gehören zum Beispiel Flyer, Visitenkarten, Kalender, Magazine und viele weitere Artikel. FA betreibt den Onlineshop und ist für die Datenannahme, Bearbeitung der Kundenaufträge und das Erstellen von Sammelbögen zuständig. Ihre Tochtergesellschaft, FLYERALARM Industrial Print GmbH (FAIP), übernimmt neben dem Drucken im *Sammeldruckverfahren* auch die Weiterverarbeitung, Montage, Konfektionierung und die Vorbereitung für den Versand. Sie ist mit acht Standorten in Deutschland vertreten und beschäftigt ca. 1200 MA. Das Kerngeschäft der IT-Abteilung ist hierbei das Aufarbeiten der Druckdaten von FA, damit diese den MA in der Produktion und den Druckmaschinen zur Verfügung gestellt werden können. Hier absolviert der Autor seine Ausbildung.

1.2 Projektbeschreibung

Durch die Änderung des Verpackungsgesetzes gibt es gesetzliche Anforderungen zur Erfassung und Meldung verwendeter Verpackungen. Die bisher verwendete Methode ist durch unterschiedliche Berechnungen und Ausnahmen an den verschiedenen Standorten nicht mehr praxistauglich. Um von den Erfahrungswerten der verpackenden MA zu profitieren, sollen die Pakete nach dem Verpacken und kurz vor dem Versand gescannt und die Abmessungen erfasst werden. Die gesammelten Daten werden mit bekannten Kartonagen-Abmessungen verglichen und in einer Datenbank abgespeichert. So können aus diesen gesammelten Daten einheitliche Verpackungsrichtlinien geschaffen und die Datenmeldung bzgl. des Verpackungsgesetzes erleichtert werden.

~~Durch die Umsetzung des Projektes können die Daten in Zukunft zuverlässiger erfasst und ausgewertet werden. Durch das automatische Erfassen wird ein großer Teil des Wartungsaufwandes weggrationalisiert und es wird sichergestellt, dass die erfassten Verpackungen aktuell und produktionsnah sind. Dadurch ist es möglich, dem neuen Verpackungsgesetz im vollen Maße gerecht zu werden und die genaue Menge der verwendeten Verpackungen zu bestimmen, wodurch Kosten gespart werden können. Auch kann so eine bessere Qualitätskontrolle gewährleistet werden, da eventuelle Schäden besser nachvollzogen werden können. Ebenso kann nachvollzogen werden, welche Verpackungsgrößen in welche Länder geliefert werden.~~

1.3 Projektschnittstellen

Die einzelnen Softwarekomponenten des Projekts kommunizieren mittels *RabbitMQ*, einer Open Source Message Broker Software, miteinander. Diese Software ist als Container im *Docker Swarm* des Unternehmens bereits vorhanden.

Ebenso werden die standortspezifischen Daten über eine bereits vorhandene Stammdatenpflege erstellt und aktuell gehalten. Diese Daten werden genutzt, um die Berechnung der Versandverpackungen zu ermitteln.

Ein Entwurf des Sensorträgers wurde anhand der Anforderungen erstellt. Die endgültige Konstruktion, das Beschaffen der Materialien sowie der Aufbau wurde unter kontinuierlicher Rücksprache von den Haustechnikern des Standorts Kesselsdorf durchgeführt.

1.4 Projektabgrenzung

Das Projekt soll als Konzept für alle anderen Standorte erstellt und umgesetzt werden. Damit sollen die Abmessungen der Pakete erfasst und einer Liefer- oder Sendungsnummer zugewiesen werden. Nicht Teil des Projekts ist es, die gemessenen Abmessungen den bekannten Abmessungen der zur Verfügung stehenden Verpackungen und bekannten Bestellungen zuzuordnen. Auch das Aufsetzen und Anlegen der Datenbank und des Datenbankbenutzers wie auch des *RabbitMQ*-Services und *Docker Swarm* sind nicht Teil des Projektes. Ebenso das Einrichten des Windowsrechners ~~ist nicht~~ Teil dieses Projekts. Diese Ressourcen waren bereits vorhanden oder wurden zur Verfügung gestellt.

2 Projektplanung

2.1 Projektphasen

Das Projekt entsteht innerhalb von 80 Stunden und ist grob in die in Tabelle 1 zu sehenden Phasen eingeteilt. Eine detaillierte Zeitplanung ist im Anhang ?? auf Seite ?? zu finden.

Tabelle 1: Grobe Zeitplanung

Projektphase	Geplante Zeit in Stunden
Analyse	10
Entwurf	8
Implementierung	39
Test	5
Einführung	6
Dokumentation	12

2.2 Abweichungen vom Projektantrag

Im Projektantrag wurde im Unterpunkt „Projektdurchführung“ noch von einem Python Sensor- und Bilderfassungs-/verarbeitungsprogramm und einer ASP .NET Core 6.0 Web API ausgegangen. Ersteres sollte die Sensor- und Bilddaten erfassen, die Abmessungen berechnen und Details aus dem Bild erkennen. Diese ausgelesenen und interpretierten Daten sollten dann von der Web API entgegengenommen und **abspeichern** werden. Kurz vor Projektstart wurde realisiert, dass eine *Microservices-Architektur* für dieses Projekt besser geeignet ist. So wird der Flaschenhals der Bilderkennung am Förderband selbst beseitigt und die ressourcenintensive Bilderkennung kann je nach Anforderung und Last auf mehrere Services und somit mehrere Docker-Container verteilt werden. Der neue Aufbau ist in Abschnitt 4 auf Seite 6 zu finden.

2.3 Ressourcenplanung

Die bereits vorhandene verwendete Hard- und Software kann dem Anhang ?? auf Seite ?? entnommen werden. Auf die zusätzlich bezogene Hardware wird in Abschnitt 3.4 auf Seite 5 eingegangen.

2.4 Entwicklungsprozess

Das Projekt verfolgt einen agilen Entwicklungsprozess, nämlich Scrum, dem Standardprozess innerhalb des IT-Teams des Unternehmens. So kann auf die sich ändernden Anforderungen eingegangen werden, die sich durch eine kontinuierliche Rücksprache mit Teamleiter, Schichtleiter der Versandanlage und der Haustechniker oder die verwendete Hardware und deren Aufbau ergeben. Ein Beispiel eines Tickets ist im ?? auf Seite ?? zu sehen.

3 Analyse

3.1 Ist-Analyse

Bisher wird die Paketgröße und Anzahl für jeden Auftrag individuell bei Eingang der Daten berechnet. Berücksichtigt werden dabei unzählige Faktoren wie beispielsweise Auflage, Stärke, Größe und Gewicht des Papiers und des Kartons, das Maximalgewicht des Paketdienstleisters und ob ein Rollenkern verwendet werden kann. Diese Berechnung ist auch für jeden Standort unterschiedlich, da nicht alle Kartonagen in jedem Werk verwendet werden. All diese in MSSQL-Datenbanken gespeicherten Informationen und diverse Sonderfälle für verschiedene Produkte müssen regelmäßig manuell gepflegt werden. Durch die Änderung des Verpackungsgesetzes gibt es nun zusätzliche gesetzliche Anforderungen an die Erfassung und Meldung verwendeter Verpackungen, was aktuell durch die unterschiedlichen Berechnungen und Ausnahmen an den Standorten nur schwer abbildbar oder planbar ist.

3.2 Soll-Analyse

Ein Paket und dessen Shipping-Label soll automatisch auf der Versandstrasse erkannt und dessen Abmessungen erfasst werden. Diese Daten sollen mit einem Bild des Pakets persistent gespeichert werden. Diese Anforderungen sollen mit Hilfe einer *Microservices-Architektur* umgesetzt werden.

3.3 „Make or buy“-Analyse

Das Projekt hat keine Gewinnabsicht sondern dient der Erfüllung der Richtlinien des Verpackungsgesetzes. Deshalb wird auf eine Amortisationsrechnung verzichtet und nur eine „Make or buy“-Analyse durchgeführt.

Auf Anfragen wurde von der Elektro Löther GmbH das in Abb. 1 auf Seite i zu sehende Angebot geschickt. In diesem rechnet die Firma Löther mit etwa 25 000 € pro Standort für Kamerahardware, Elektrik/Mechanik, Software und Inbetriebnahme.

Die Kosten einer Eigenentwicklung mit der in Abschnitt 3.4 auf der nächsten Seite ausgewählten Hardware sind in Tabelle 2 zu sehen. Die Personalkosten sind in Tabelle 3 auf der nächsten Seite dargestellt. Der Stundenlohn des Personals wurde von der Personalabteilung als Orientierungswert so kommuniziert. Zu diesem kommen die Ressourcenkosten, zu denen zum Beispiel Strom und Miete der Büros zählen. Dafür wird ein Pauschalbetrag pro Stunde von 15 € verwendet.

Tabelle 2: Kostenverteilung Hardware

Hardware	Stück / Meter	Kosten pro Stück / Meter	Gesamt
ARCELI Shield Board Kit	1	17,99 €	17,99 €
AZDelivery 5 x Mega 2560 R3	1	74,99 €	14,99 €
Benewake TF MINI PLUS	3	61,80 €	182,40 €
Microsoft Lifecam Studio	1	42,99 €	42,99 €
item-Systemprofile	4	7,38 €	29,52 €
item-Verbindungsstücke	24	8,00 €	192,00 €
item-Füße	2	24,00 €	48,00 €
Dell Wyse 5070 Thin Client	1	450,00 €	450,00 €
Gesamtkosten			977,89 €

Die Gesamtkosten einer Eigenentwicklung betragen 3095,89 €, was etwa ein Zehntel der von der Firma Löther geschätzten Kosten für einen Standort sind. Für jeden weiteren Standort fallen im Fall der Eigenentwicklung nur die Kosten für die Hardware und die Haustechniker an. Aufgrund dieses großen Unterschieds und der daraus entstehenden Einsparung von mehr als 20 000 € pro Standort wurde sich für die Eigenentwicklung entschieden.

Tabelle 3: Kostenverteilung Personal

Personal	Zeit in Stunden	Kosten pro Stunde	Gesamt
Auszubildender	80	6,00 € + 15,00 €	1680,00 €
Teamleitung	2	31,50 € + 15,00 €	93,00 €
Teammitglied	2	21,50 € + 15,00 €	73,00 €
Haustechnik	8	19,00 € + 15,00 €	272,00 €
Gesamtkosten			2118,00 €

3.4 Analyse der benötigten Hardware

Um die Abmessungen der Verpackungen hardwaretechnisch erkennen zu können, werden Sensoren benötigt. Es stehen drei verschiedene Abstandssensoren zur Verfügung, die im Folgenden auf ihre Nutzbarkeit überprüft werden. Von den Sensoren werden im Abstand von fünf bis 55 cm alle fünf Zentimeter 60 s lang die Messwerte aufgezeichnet. Aus diesen Messreihen wird die Tauglichkeit der Sensoren entnommen.

Infrarot (IR)-Sensor Die Auswertung für diesen Sensor ist in Abb. 2 auf Seite i zu sehen. Das Driften der Messwerte ist ein Ausschlusskriterium, weswegen der Sensor nicht berücksichtigt werden kann.

Ultraschallsensor Die Auswertung für diesen Sensor ist in Abb. 3 auf Seite ii zu sehen. Das Jittern der Messwerte ist ein Ausschlusskriterium, weswegen der Sensor nicht berücksichtigt werden kann.

Lasersensor Die Auswertung für diesen Sensor ist in Abb. 4 auf Seite ii zu sehen. Die Genauigkeit dieses Sensors war ausreichend für diese Anwendung, weswegen die Entscheidung auf diesen Sensor fiel.

Als Kamera wird vorerst aus Kostengründen eine bereits vorhandene Webcam, eine Microsoft Lifecam Studio, verwendet. Diese reichte mit einer 1080p Full HD-Auflösung in den ersten Tests vollkommen aus. Als Ersatzlösung steht der KEYENCE SR-X100 AI-Codeleser zur Verfügung, der allerdings mit über 1500 € für dieses Konzept zu teuer ist. Anzumerken ist dennoch, dass die Umsetzung des Projekts selbst mit der Beschaffung eines solchen Codeleser deutlich günstiger als das Angebot der Firma Löther ist.

3.5 Analyse des Standorts

Die Pakete sollen am Versandband ausgemessen werden. Da das Projekt auch an anderen Standorten umgesetzt werden soll, müssen einige Punkte beachtet werden:

- Für den Aufbau des Sensorträgers muss entsprechend Platz sein.
- Die Anschlüsse für Strom und Netzwerk müssen gut erreichbar sein.

- Das Vermessen soll nach dem Anbringen des Versandlabels erfolgen.
- Die Förderstrecke soll möglichst gerade verlaufen.

Eine grobe Skizze des entsprechenden Abschnitts der Versandanlage in Kesselsdorf der als Aufbauort nach Rücksprache mit dem Teamleiter der IT, der Haustechnik und dem Schichtleiter der Versandanlage festgelegt wurde, ist in ?? auf Seite ?? zu sehen.

4 Entwurf

4.1 Zielplattform

Das Projekt besteht aus vier *Microservices*. Diese sind in den Programmiersprachen C, C# und Python umgesetzt worden. Für die Datenspeicherung wird eine MSSQL-Datenbank eingesetzt, da die gut strukturierten und gleichbleibenden Daten sehr gut zu einer relationalen Datenbank passen. Bereitgestellt wird die Datenbank von einem Windowsserver. Das Auslesen der Daten findet auf einem Einplatinencomputer statt, der die Sensordaten über die Serial-Schnittstelle an einen Windowsrechner überträgt. Auf diesem Windowsrechner analysiert ein Python-Programm die Daten und verknüpft sie mit einem Kamerabild. Die Speicherung der Daten erfolgt mit einer unter Verwendung von .NET 6.0 erstellten Konsolen-Anwendung. Die Berechnung der Paketgröße und das Erkennen des Pakets und der Sendungsnummer erfolgt mit einem Python-Programm und mit den populären Programmbibliotheken *OpenCV* und *YOLOv7*. Die Kommunikation zwischen den Services erfolgt über *RabbitMQ*. Die Konsolen-Anwendung, die Berechnung der Paketgröße und Erkennung des Pakets sowie *RabbitMQ* werden auf einem *Docker Swarm* bereit gestellt. Diese sind auch über die interne GitLab Continuous Integration/Continuous Delivery (CI/CD)-Pipeline angebunden, was das Pflegen und Deployment der Dienste automatisiert und vereinfacht.

4.2 Datenmodell

Die drei Datenbankmodelle sind sehr einfach, da relativ wenig Daten gespeichert werden und eine geringe Abhängigkeit vorhanden ist. Das Tabellenmodell der Tabellen ist in Abb. 5 auf Seite iii zu sehen.

4.3 Architekturdesign

Als Anwendungsarchitektur wurde die *Microservices-Architektur* gewählt. Anstatt die Software als einzelne Einheit aufzubauen, wird bei einer *Microservices-Architektur* der modulare Ansatz verfolgt. Diese können unabhängig voneinander deployed, gewartet und verwaltet werden. Von der losen Kopplung, dem separaten Deployment und der hohen Skalierbarkeit kann vor allem die ressourcenintensive Bilderkennung profitieren. In ?? auf Seite ?? ist die Architektur als Skizze zu

sehen. Die Kommunikation der Services untereinander erfolgt über *RabbitMQ*, ein Open Source Message Broker, der das Advanced Message Queuing Protocol (AMQP) implementiert.

4.4 Entwurf des Sensorträgers

Die Kamera, die drei Lasersensoren, der Arduino ~~als auch~~ der Windowsrechner sollen am Rollenförderband an einem Träger montiert werden. Dazu wurde nach der Analyse der benötigten Hardware und des Standorts, in Abschnitt 3.4 auf Seite 5 und Abschnitt 3.5 auf Seite 5 beschrieben, unter Rücksprache mit dem Schichtleiter des Versands und einem Techniker der Haustechnik ein Entwurf für einen Sensorträger erstellt. Der 2D-Entwurf ist in Abb. 7 auf Seite v, der 3D-Entwurf ist in Abb. 8 auf Seite vi zu sehen. Die seitlichen Sensoren sind nur wenige Zentimeter über dem Rollenförderband angebracht, sodass auch sehr niedrige Pakete erfasst werden können. Der Höhen-Sensor ist mittig angebracht, da der Sensorträger kurz nach der Maschine für das Anbringen des Versandlabels montiert werden soll, welche die Pakete auf dem Förderband zentriert. Die Kamera sitzt ebenfalls mittig, um ein Paket in voller Größe aufnehmen zu können.

Die Sensoren selbst haben nur winzige Aufnahmen für kleine Gewindestangen als Befestigungs-
option. Um etwas mehr Schutz vor Stoßschäden und eine bessere ~~Option der Befestigung~~ zu
haben, wurden die Sensorhalterungen 3D gedruckt. Dazu wurde die Vorlage von Thingiverse [3]
verwendet. Die Vorlage und das zusammengesetzte Endprodukt ist in Abb. 6 auf Seite iv zu
sehen.

5 Implementierungsphase

5.1 Erstellung der Datenbanktabellen

Nach der Analyse wurde das in Abb. 5 auf Seite iii zu sehende Tabellenmodell erstellt und auf Basis dessen die jeweiligen Tabellen. Als Beispiel für die Erstellung einer Tabelle ist das passende SQL-Skript für die RawData-Tabelle in Listing 4 auf Seite xiii zu sehen.

5.2 Erstellen der RabbitMQ Exchanges

Um die Kommunikation zwischen den Services zu ermöglichen, wurden zwei *RabbitMQ*-Exchanges erstellt. Hier können Services als Publisher Nachrichten in eine oder mehrere Queues einreihen und Subscriber können diese Nachrichten dann abrufen. Es wurde sich jeweils für einen Fanout-Exchange entschieden, einen für die un- und einen für die verarbeiteten Daten. Ein Fanout-Exchange verteilt die eingegangenen Nachrichten auf alle ihm bekannten Queues. In diesem Fall ist das sinnvoll, da es bei beiden Exchanges sein kann, dass in Zukunft noch andere Services auf die Daten zugreifen sollen. Für beide Exchanges wurde jeweils für den Publisher und den Subscriber eine Authentifikation in Form von Nutzernamen und Passwort angelegt.

5.3 Erstellen des Datenspeicherservice

5.3.1 Verwendete Pakete

Für die Anbindung der Anwendung an *RabbitMQ* wurde das intern entwickelte und bereitgestellte NuGet-Paket FAIP.LIB.RMQ verwendet. Zur Anbindung der SQL-Datenbank (`Microsoft.EntityFrameworkCore.SqlServer`) und zum Erstellen der Modelle aus den Tabellen der Datenbank (`Microsoft.EntityFrameworkCore.Tools`) wurde das NuGet-Paket Entity Framework Core angewandt. Für das Loggen wichtiger Informationen und Fehlermeldung wurde das NuGet-Paket NLog installiert.

5.3.2 Datenbankanbindung

Die Datenbankanbindung erfolgte mit Hilfe der `Microsoft.EntityFrameworkCore.Tools`. Dieses bietet bei der Verwendung des „Database first“-Ansatzes die Möglichkeit, die Datenbanktabellen als Modelle in die Anwendung zu laden. Die beiden Modelle für `RawData` und `ProcessedData` sowie der `DbContext`, der die Verbindungsschnittstelle zwischen Code und Datenbank bildet, wurde mit dem in Listing 1 zu sehenden Befehl erstellt.

Listing 1: Portierungsbefehl der Datenbanktabellen zu C#-Modellen

```
1 Scaffold-DbContext "Data Source=sql-mar-01.druckhaus.local; Initial Catalog=PAMESAN; persist security info=True; user id=pamesan-rw; password=*****"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir DatabaseContext -Tables RawData, ProcessedData
```

5.4 Platzierung der Sensoren und der Webcam

Zum Abmessen des Pakets wurden drei Lasersensoren an item-Aluprofilen angebracht, als 3D-Zeichnung in Abb. 8 auf Seite vi und als fertiger Sensorträger in Abb. 9 auf Seite vii zu sehen. Die Höhe wird beim Durchlaufen durch die Differenz zwischen bekanntem Abstand zwischen Förderband und Sensor und gemessenen Abstand Paket und Sensor bestimmt. Die Breite wird beim Durchlaufen durch die Differenz zwischen bekanntem Abstand zwischen linkem und rechten Sensor und den beiden gemessenen Abständen zum Paket bestimmt. Um die Länge l_P des Pakets bestimmen zu können, muss die Bandgeschwindigkeit v_B berechnet werden. Diese kann je nach Auslastung des Förderbandes und der Produktion unterschiedlich sein. Die beiden seitlich angebrachten Sensoren, s_1 und s_2 , sind mit einem bekannten Abstand a voneinander versetzt angebracht. Es wird jeweils der erste t_1 und letzte t_2 Zeitpunkt erfasst, an dem ein Sensor einen Abstandswert gemessen hat. Aus dem bekannten Abstand zwischen linken und rechten Abstand a und der Differenz der Erfassungszeitpunkte Zeit kann nun die Bandgeschwindigkeit v_B , siehe Gleichung (1) auf der nächsten Seite, berechnet werden. Mit dieser Geschwindigkeit v_B und der

Differenz zwischen dem ersten und dem letzten Erfassungszeitpunkt eines Sensors, $s1_{t2}$ und $s1_{t1}$, kann nun die Länge l_P berechnet werden, siehe Gleichung (2).

$$v_B = \frac{a}{s2_{t1} - s1_{t1}} \quad (1)$$

$$l_P = (s1_{t2} - s1_{t1}) \cdot v \quad (2)$$

Die Webcam wird überhalb des Förderband etwas nach hinten versetzt montiert, um ein vollständiges und gerades Bild vom Paket zu haben.

5.5 Erstellen der Sensordaten-Abfrage-Anwendung

5.5.1 Verwendete Pakete

Zum Auslesen der Sensordaten mittels des Arduino-Einplatinencomputers wurde die SoftwareSerial-Bibliothek installiert, die das Auslesen von verschiedenen Pins auf dem Board erlaubt. Neben den eingebauten Python-Modulen wurde zum Erfassen der Sensor- und Bilddaten opencv-python, die Implementierung von *OpenCV* in Python und pyserial sowie zur Kommunikation mit *RabbitMQ* das Paket pika verwendet.

5.5.2 Implementierung der Ausleselogik der Sensor- und Bilddaten

Die drei Lasersensoren sind mit einem Arduino verbunden, zu sehen in Abb. 10 auf Seite viii. Für das Auslesen von mehreren Benewake TF MINI PLUS-Sensoren stellt der Hersteller ein Skript zur Verfügung, das für zwei Sensoren ausgelegt ist. [6] Dieses Skript wurde so angepasst, dass es mit drei Sensoren arbeiten kann und die jeweils gemessenen Abstände sowie einen Millisekunden-Zeitstempel über die serielle Schnittstelle ausgibt.

Das Python-Programm liest beim Starten zuerst die Umgebungsvariablen aus, um Baudrate und ComPort des Arduinos sowie den Standort und die Verbindungsparameter zu *RabbitMQ* zu ermitteln. Danach wird die Verbindung mit dem Arduino über die serielle Schnittstelle initialisiert. Nach erfolgreichem Initialisieren werden aus den ersten zehn gemessenen Abständen jedes Lasersensors der jeweilige Maximalwert und abzüglich einer Toleranz damit der jeweilige Schwellwert ermittelt. Danach ist das Auslesen, die Datenverarbeitung und die Kommunikation mit *RabbitMQ* in je einen eigenen Prozess ausgelagert. Diese Prozesse sind mit Queues miteinander verbunden, so dass ein Datenaustausch stattfinden kann.

Der Auslese-Prozess liest den Datenstring der seriellen Schnittstelle aus und wandelt diesen zu einem Dictionary um. Fällt der Abstandsmesswert des ersten seitlichen Sensors unter den zuvor ermittelten Schwellwert, wird das zugehörige Dictionary und alle kommenden Messwerte als Liste gepflegt, bis der Schwellwert wieder überschritten wird. In diesem Fall wird der

Datenverarbeitungsprozess über die entsprechende Queue informiert, ein Bild zwischenspeichern. Sind alle Schwellwerte der Sensoren wieder überschritten, wird die Liste über die für die Datenverarbeitung vorgesehene Queue an den Datenverarbeitungsprozess übergeben.

Der Datenverarbeitungsprozess verarbeitet mit Hilfe von opencv-python den Kamerastream und speichert das aktuelle Bild zwischen, wenn die entsprechende Anweisung durch über die Queue erfolgt. Befindet sich eine Liste mit Werten in der Queue, wird für jeden Sensor der jeweils am häufigsten auftretende Wert ermittelt, sowie dessen erstes und letztes Auftreten inklusive des passenden Zeitstempels. Diese Werte werden gemeinsam mit dem als base64-String kodierten Bild an den Kommunikationsprozess weitergereicht.

Dieser erstellt aus den übertragenen Werten, dem Bild, dem aus den Umgebungsvariablen ausgelesenen Standort, dem aktuellen Zeitpunkt und einer generierten Universally Unique Identifier (UUID) ein JSON-Objekt, das mit Hilfe von pika an den RawData-Exchange versendet wird.

5.6 Erstellen des Datenverarbeitungsservice

5.6.1 Verwendete Pakete

Auch im Datenverarbeitungsservice wurde pika zur Kommunikation mit *RabbitMQ* sowie opencv-python, numpy und pyzbar zum Erkennen und Auslesen der Barcodes verwendet. Als Objektdetektor zum Erkennen der Versandverpackungen wurde ein durch *YOLOv7* bereitgestelltes und mit eigenen Bildern trainierten *Fully Convolutional Neural Network* (FCNN) verwendet. Für das Abfragen der Datenbank wurde SQLModel eingesetzt. Zum Labeln der Bilder für die eigenen Trainingsdaten wurde labelImg verwendet, ein Werkzeug für grafische Bildanmerkungen. [10]

5.6.2 Zusammenspiel der einzelnen Komponenten

Der Kommunikationsdienst schaltet sich als Subscriber auf den RawData-Exchange auf. So liest dieser alle ankommenden Daten aus leitet diese an den Erkennungsprozess für Versandverpackungen weiter.

Dieser erkennt das Paket im Bild, berechnet anhand dieser Daten ein Rechteck um das Paket, schneidet das Bild entsprechend zu und leitet dieses an den Label-Auslese-Prozess weiter. Hier wird der Barcode im Label auf dem Paket erkannt und ausgelesen. Diese Daten werden dann mit der berechneten Paketgröße zusammengeführt und über den Kommunikationsprozess an den ProcessedData-Exchange gesendet.

5.6.3 Erkennung der Versandverpackung

Ursprünglich sollte die Erkennung der Versandverpackungen mittels opencv-python erfolgen. Im Laufe der Implementierung wurde allerdings festgestellt, dass die Spiegelung der Metallrollen des Rollenförderbands eine verlässliche Erkennung des Pakets mittels opencv-python unmöglich machen. Abhilfe hat in diesem Fall *YOLOv7* geschaffen. *YOLOv7*, ein *Fully Convolutional Neural Network*, ist ein *Deep Learning*-Objektdetektor, der Objekte wie Menschen, Autos, Katzen und viele weitere durch *Semantische Segmentierung* erkennen kann. Durch die Implementierung sogenannter Bag of Freebies (BoF), Methoden, die die Performance eines Modells erhöhen, ohne die Trainingszeit zu verlängern, erlaubt *YOLOv7* das Trainieren eines eigenen Modells mit einem relativ kleinen Datensatz. [8] Das Trainieren des eigenen Modells wird mit Hilfe von *Transfer Learning* ermöglicht, wobei ein bereits trainiertes Modell verwendet und mittels eigener Daten angepasst wird. *Transfer Learning* erlaubt eine schnellere Erstellung, eine gute Modellqualität und weniger Ressourceneinsatz. [12]

Für das eigene Modell wurden 100 Fotos von Versandverpackungen aufgenommen und mit labellImg gelabelt. Ein Beispiel einer für den Label-Prozess ist in Abb. 11 auf Seite ix zu sehen. 70 der Bilder und Label wurden als Trainingsdaten und der Rest als Daten zur Validierung verwendet. Mit der auf GitHub veröffentlichten Implementierung von *YOLOv7*, den 70 Trainingsbildern und der in Listing 2 zu sehenden Anweisung wurde das Training des eigenen Modells gestartet.

Listing 2: Befehl zum Trainieren des Modells

```
1 python train.py --device 0 --batch-size 16 --epochs 100 --img 640 640 --data
  data/custom_data.yaml --hyp data/hyp.scratch.custom.yaml --cfg cfg/training/
  yolov7_custom.yaml --weights yolov7.pt --name yolo7-custom
```

Die durch das Training erstellten Gewichte (weights) erlauben das Erkennen von den zuvor ausgewählten und gelabelten Objekten ähnlicher Art. Ein Beispiel dessen Ausführung ist in Abb. 13 auf Seite x zu sehen.

Mit dem Aufruf der Detection-Klasse werden die Gewichte geladen, das Modell vorbereitet sowie alle möglichen Labels (in diesem Fall nur eins) geladen. Wird die detect-Methode mit einem Bild aufgerufen, wird versucht, ein Paket in diesem Bild zu erkennen. Bei Erfolg wird ein Rechteck um das Objekt gezeichnet sowie das Label und die Zuversichtlichkeit an dieses geschrieben. Dieses Bild wird zum Speichern in der Datenbank behalten. Für die Weiterverarbeitung wie dem Erkennen und Erfassen des Barcodes auf dem Label, wird das Bild an dem erkannten Rechteck zugeschnitten und an die BarcodeDetection-Klasse weitergeleitet. Wird kein Paket im Bild erkannt, bricht die Methode den Erkennungsprozess frühzeitig ab, es wird ein Null-Wert als Bild dem Kommunikationsprozess übergeben und das Ereignis wird geloggt.

5.6.4 Auslesen des Versandlabels

Das Auslesen des Versandlabels erfolgt mit opencv-python und numpy. Dazu wird das Bild in ein Graustufen-Bild umgewandelt und diverse Filter darüber gelegt. Da ein Barcode typischerweise schwarz-weiß ist, werden so beispielsweise nur hohe und niedrige Schwellwerte beibehalten. Danach wird nach Konturen gesucht, die einem Barcode ähneln. Mittels eines Blur-Filters wird das Rauschen im Bild entfernt. Durch mehrfache Erosion kann ein großes Rechteck an der Stelle ausgemacht werden, an der ein Barcode vorhanden ist. Diese Stelle im Bild wird als Rechteck markiert, zu sehen in Abb. 14 auf Seite xi. Im Falle, dass mehrere Rechtecke vorhanden sind, wird das mit der größten Kontur ausgewählt. Anhand der Koordinaten des Rechtecks wird ein Bildausschnitt erstellt, zu sehen in Abb. 12 auf Seite ix, der nun mittels pyzbar ausgelesen werden kann.

5.6.5 Berechnung der Paketgröße

Zum Berechnen der Paketgröße werden jede Stunde die standortspezifischen Daten aus der Datenbank abgefragt. In diesen ist die SiteId, die Distanz zwischen den beiden gegenüberliegenden Sensoren, die Distanz zwischen dem auf das Band schauende Sensor und dem Förderband, der horizontale Abstand zwischen den beiden gegenüberliegenden Sensoren sowie eine Beschreibung enthalten. Dazu wurde mit Hilfe von SQLModel eine Tabellenklasse erstellt, die die Datenbanktabelle widerspiegelt, sowie eine Verbindungsmethode, die sich um die Verbindung zur Datenbank kümmert.

Die Berechnung erfolgt wie in Abschnitt 5.4 auf Seite 8 beschrieben. Je eine Methode ist für die Berechnung für jeweils Länge, Breite und Höhe zuständig.

5.7 Bestückung des Sensorträgers

Nachdem die Haustechnik den Sensorträger gebaut hatte, musste dieser mit Sensoren, Kamera, Arduino und Windowsrechner bestückt sowie alle Kabel ordentlich verlegt werden. Dazu wurden die Sensoren mit 4-adrig verschraubbaren Sensorkabeln versehen und in die 3D-gedruckten Sensorhalterungen gesteckt. Mittels item-Verbindungsstücken wurden Sensoren, Kamera, Rechnerkäfig und ABS Kunststoff Gehäuse an den item-Aluprofilen befestigt. Der Rechner wurde in den Rechnerkäfig, der Arduino in dem ABS Kunststoff Gehäuse angebracht. Die Kabel wurden im Kabelkanal ordentlich verlegt, die Kamera mit dem Rechner verbunden und die Sensoren mit den am Arduino angebrachten Female-Steckern verschraubt.

6 Abnahme- und Einführungsphase

6.1 Code-Review

Während der Erstellung des Projekts wurde immer wieder Rücksprache mit anderen Programmierern des Unternehmens gehalten sowie deren Anregungen eingeholt. Jeder Service wurde nach Fertigstellung gemeinsam mit einem erfahrenen Anwendungsentwickler auf Fehler oder Verbesserungen überprüft. So wurde der Datenspeicherservice ursprünglich als zwei Services geplant. Durch die große Ähnlichkeit beider wurde aber durch den Review-Prozess beispielsweise entschieden, diese Services als eine Anwendung umzusetzen. Beide Services beziehen ihre Verbindungsdaten zu den *RabbitMQ*-Exchanges und ihren damit verbundenen Namen aus den Umgebungsvariablen, die durch die docker-compose-Dateien gesetzt werden. Deployed werden beide Services dann automatisch durch eine CI/CD-Pipeline, die für die Ausführung verantwortliche Datei ist in Listing 3 auf Seite xii zu sehen. Durch können zwei Services aus einer Anwendung heraus gestartet werden. Auch wurden durch den Review-Prozess die Berechnungen der Länge, Breite und Höhe, zuvor noch als eine Methode, in einzelne Methoden aufgegliedert, um die Übersichtlichkeit zu erhöhen. Kurz vor der Inbetriebnahme fiel noch auf, dass das aufzuzeichnende Kamerabild nicht als byte-array in ein JSON-Objekt eingefügt werden kann, da JSON keine byte-arrays verarbeiten kann. Deshalb wurde hier das byte-array zu einem UTF8-String umgewandelt.

6.2 Inbetriebnahme und Abnahme

Die Inbetriebnahme erfolgte am 26.10.2022 mit der Montage des Sensorträgers am Rollenförderband. Zuvor wurden der Datenspeicherservice sowie der Datenverarbeitungsservice auf dem *Docker Swarm* deployed, sowie der Windowsrechner mit der Datenauslesesoftware und der Arduino mit der Sensorsauslesesoftware ausgestattet. Nach einigen Feinjustierungen der seitlichen Sensoren bzgl. der Höhe und dem Ausrichten der Kamera konnten die ersten Daten empfangen werden. Nach dem Sammeln von Daten für einen Tag wurde das Projekt vom Teamleiter der IT abgenommen. Dies erfolgte durch die Vorstellung aller Komponenten, der Begutachtung des Sensorträgers sowie der Betrachtung der bereits gesammelten Daten. Anschließend wurde über mögliche Verbesserungen sowie das weitere Vorgehen bzgl. der Verarbeitung der Daten gesprochen.

7 Dokumentation

Für alle Services wurde eine Entwicklerdokumentation erstellt. Diese ist in Anhang B auf Seite B-1 zu finden. Bei der Dokumentation wurde vor allem Wert gelegt, Projekt- und Service-spezifische Punkte zu dokumentieren, sodass das Pflegen der bestehenden Anlage und Anwendungen sowie eine Replikation möglich ist. So wird die Einbindung der intern entwickelten

RabbitMQ-Bibliothek nicht erklärt, da es dafür bereits eine Anleitung gibt, dafür wird auf das Trainieren eines neuen Modells oder den Aufbau des Sensorträgers ~~genauer eingegangen~~. Insgesamt soll die Dokumentation einen groben Überblick über die Funktionen und Herangehensweise der Services geben. Hyperlinks des internen Speichers, GitLab- und Confluence-Systems sind für diese Dokumentation angepasst, um personenbezogene Daten zu schützen. Intern bekannte Abkürzungen werden nicht genauer erklärt.

8 Fazit

8.1 Soll-/Ist-Vergleich

Das Projekt wurde zur Zufriedenheit des Versands und der IT erfolgreich abgeschlossen. Das Umschwenken auf eine *Microservices-Architektur* kurz vor Projektbeginn hat keine Probleme verursacht. Während der Planung, Analyse und Umsetzung traten bis auf das Problem mit der Bilderkennung keine größeren oder unerwarteten Probleme auf. Dieses Problem konnte jedoch durch den Einsatz von *YOLOv7* gelöst werden, was jedoch ~~die Folge eines verschobenen Zeitplans~~ hatte, wodurch der Schreibtischtest gestrichen und die Abnahme verkürzt werden musste, um in den vorgegebenen 80 Stunden Projektzeit zu bleiben. Der dadurch entstandene Zeitablauf ist in Tabelle 4 zu sehen.

Tabelle 4: Abschließender Zeitablauf

Projektphase	Geplant	Tatsächlich	Differenz
Analyse	10 h	10 h	
Entwurf	8 h	8 h	
Implementierung	39 h	46 h	7 h
Test	5 h	2 h	-3 h
Einführung / Abnahme	6 h	2 h	-4 h
Dokumentation	12 h	12 h	

Fünf Tage nach Abnahme hat die eingesetzte Webcam leider nicht mehr funktioniert. Eine genaue Ursache ist nicht bekannt, vermutlich hat die andauernde Nutzung zu einer Überbelastung geführt. Das weitere Vorgehen ist in Abschnitt 8.3 auf der nächsten Seite beschrieben.

8.2 Lessons Learned

Es gab drei große Punkte, die aus diesem Projekt mitgenommen werden können:

Pufferzeit Für das nächste Projekt sollten etwa ein bis zwei Prozent der Gesamtzeit als Pufferzeit eingeplant werden, um auf Probleme besser reagieren zu können.

YOLOv7 Als Objektdetektor hat sich *YOLOv7* als extrem praktisch und leicht einsetzbar profiliert. Auch mit geringen Vorkenntnissen kann dieser mit eigenen Objekten trainiert und eingesetzt werden.

Webcam Es hat sich herausgestellt, dass die andauernde Nutzung einer Webcam kein guter Ersatz für eine Industriekamera ist. In Zukunft wird hier auf entsprechende Modelle gesetzt.

8.3 Ausblick

Als Konzept hat dieses Projekt gezeigt, dass das automatische Erkennen und Ausmessen der Kartonagen zur Erfüllung der Novelle des Verpackungsgesetzes umsetzbar ist. Die gesammelten Daten müssen allerdings noch aufbereitet, ausgewertet und für das Weiterreichen angepasst werden. Der Sensorträger soll wie in diesem Projekt realisiert an allen anderen Standorten auch aufgebaut und angeschlossen werden. Lediglich die Webcam wird durch den in Abschnitt 3.4 auf Seite 5 erwähnten KEYENCE SR-X100 AI-Codeleser ersetzt. In diesem Fall kann auch die Erfassung des Labels und damit das Auslesen des Barcodes ersetzt werden. **Auch** möchte die Abteilung für Qualitätsmanagement Zugriff auf die Daten haben, was durch eine Webanwendung gelöst werden soll. Das sind allerdings weitere Projekte und Anpassungen, die in Zukunft angegangen werden sollen.

Eidesstattliche Erklärung

Ich, Johannes Leyrer, versichere hiermit, dass ich meine Dokumentation zur betrieblichen Projektarbeit mit dem Thema

Entwicklung von PaMesAn — Implementierung eines neuen Systems zur Erfassung von Versandverpackungen mit Hilfe von Bild- und Sensordaten zur Erfüllung der Novelle des Verpackungsgesetzes

selbstständig verfasst und keine nicht angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Dresden, den 14.11.2022

Johannes Leyrer

Literatur

- [1] Amazon Web Services Inc. (Hg). *Was sind Microservices? / AWS.* de-DE. Verfügbar unter: <https://aws.amazon.com/de/microservices/>. abgerufen am 21.10.2022.
- [2] MathWorks (Hg). *Semantische Segmentierung.* de. Verfügbar unter: <https://de.mathworks.com/solutions/image-video-processing/semantic-segmentation.html>. abgerufen am 30.10.2022.
- [3] Thinigiverse.com (Hg). *TFmini Plus holder mount by tonyi.* en. Verfügbar unter: <https://www.thingiverse.com/thing:4592503>. abgerufen am 30.10.2022.
- [4] Wikipedia (Hg). *OpenCV.* de. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=OpenCV&oldid=226224987>. abgerufen am 21.10.2022.
- [5] Wikipedia (Hg). *RabbitMQ.* de. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=RabbitMQ&oldid=222274219>. abgerufen am 21.10.2022.
- [6] benewakesupport u. a. *TFmini_Arduino.* en. Verfügbar unter: <https://github.com/TFmini/TFmini-Arduino>. abgerufen am 24.10.2022.
- [7] LeanIX GmbH. *Microservices-Architekturen – Der ultimative Guide / LeanIX.* de. Verfügbar unter: <https://www.leanix.net/de/wiki/vsm/microservices-architecture>. abgerufen am 21.10.2022.
- [8] Sovit Rath u. a. *YOLOv7 Paper Explanation: Object Detection and YOLOv7 Pose.* en-US. Verfügbar unter: <https://learnopencv.com/yolov7-object-detection-paper-explanation-and-inference/>. abgerufen am 21.10.2022.
- [9] Ajay Sarangam. *Fully Convolutional Network (FCN): A Basic Overview In 2021.* en-US. Verfügbar unter: <https://www.jigsawacademy.com/blogs/data-science/fcn/>. abgerufen am 30.10.2022.
- [10] Darren Tzutalin u. a. *LabelImg.* en. Verfügbar unter: <https://github.com/heartexlabs/labelImg>. abgerufen am 29.10.2022.
- [11] Laurenz Wuttke. *Deep Learning: Definition, Beispiele & Frameworks.* de-DE. 04/2022. Verfügbar unter: <https://datasolut.com/was-ist-deep-learning/>. abgerufen am 30.10.2022.
- [12] Laurenz Wuttke. *Transfer Learning: Grundlagen und Einsatzgebiete.* de-DE. 04/2022. Verfügbar unter: <https://datasolut.com/was-ist-transfer-learning/>. abgerufen am 30.10.2022.

Glossar

Deep Learning, ein Teilgebiet des maschinellen Lernens, spezialisiert auf große Datenmengen. [11]. 11, V

Docker Swarm, ein Cluster aus Docker-Containern. 2, 6, 13

Fully Convolutional Neural Network, eine Architektur, die hauptsächlich für die *Semantische Segmentierung* verwendet wird. [9]. 10, 11, VI

Microservices, sind kleine Softwareprogramme, die aus unabhängigen Services bestehen und über definierte APIs kommunizieren. [1]. 6

Microservices-Architektur, zielt darauf ab, Software als kleine, einzelne Einheiten aufzubauen, die unabhängig voneinander deployed und verwaltet werden können. [7]. 3, 4, 6, 14

OpenCV, stellt als Programmbibliothek Algorithmen für die Bildverarbeitung und Computer Vision zur Verfügung. [4]. 6, 9

RabbitMQ, eine Open Source Message Broker Software, die das Advanced Message Queuing Protocol (AMQP) implementiert. [5]. 2, 6–10, 13, 14

Sammeldruckverfahren, ein Produktionsverfahren, bei dem versucht wird, Aufträge verschiedener Kunden aber gleichem Papier zusammen auf einen Bogen zu platzieren. Die Minimierung der Weißfläche senkt die Produktionskosten und damit verbunden den Preis für den Kunden. 1

Semantische Segmentierung, ist ein *Deep Learning*-Algorithmus, der jedem Pixel in einem Bild eine Bezeichnung oder Kategorie zuordnet. [2]. 11, V

Transfer Learning, eine Methode von *Deep Learning*, bei dem der Lernfortschritt eines bestehenden Modells transferiert wird. [12]. 11

YOLOv7, auch „you only look once, version 7“, stellt als Programmbibliothek einen hochmodernen Objektdetektor zur Verfügung. [8]. x, 6, 10, 11, 14, 15, VII

Abkürzungsverzeichnis

AMQP	Advanced Message Queuing Protocol	7
BoF	Bag of Freebies	11
CI/CD	Continuous Integration/Continuous Delivery	6
FA	FLYERALARM GmbH	1
FCNN	<i>Fully Convolutional Neural Network</i>	10
FAIP	FLYERALARM Industrial Print GmbH	1
IR	Infrarot	5
MA	Mitarbeitenden	1
UUID	Universally Unique Identifier	10

Abbildungsverzeichnis

1	Angebotsvergleich Fa. Löther – Kartonagenerkennung	i
2	Auswertung IR-Sensor	i
3	Auswertung Ultraschallsensor	ii
4	Auswertung Lasersensor	ii
5	Tabellenmodell	iii
6	Sensorhalterung; Vorlage [3] und zusammengesetztes Endprodukt	iv
7	Sensorträger 2D	v
8	Sensorträger 3D gezeichnet	vi
9	Fertig montierter und aufgestellter Sensorträger	vii
10	Arduino mit angeschlossenen Sensoren und Shield	viii
11	Beispiel Verwendung labelImg	ix
12	Beispiel Barcodeerfassung	ix
13	Beispiel Versandverpackungserkennung mit <i>YOLOv7</i>	x
14	Beispiel Labelerfassung	xi
15	2D-Zeichnung Sensorträger	B-2
16	3D-Zeichnung Sensorträger	B-3
17	Anschlusspins der 4-adrig verschraubbaren Male-Sensorkabel	B-3
18	Pinbelegung am Arduino	B-3

Tabellenverzeichnis

1	Große Zeitplanung	2
2	Kostenverteilung Hardware	4
3	Kostenverteilung Personal	5
4	Abschließender Zeitablauf	14

Listings

1	Portierungsbefehl der Datenbanktabellen zu C#-Modellen	8
2	Befehl zum Trainieren des Modells	11
3	.gitlab-ci.yml-Datei für CI/CD	xii
4	SQL-Skript für RawData	xiii
5	custom_data.yaml	B-9
6	Befehl zum Trainieren des Modells	B-9

A Anhang

A.1 Angebot der Elektro Löther GmbH

Angebotsvergleich Fa. Löther - Kartonagenerkennung

Standorte	Dillberg	Heuchelhof	Klipphausen	Kesselsdorf	Summe Standorte
Kostenaufteilung					
Kamerahardware	13.338,79 €	13.338,79 €	13.338,79 €	19.081,04 €	59.097,41 €
Eletrik / Mechanik	2.095,00 €	2.095,00 €	2.675,00 €	3.845,00 €	10.710,00 €
Software & IBN	5.225,00 €	2.900,00 €	5.150,00 €	6.900,00 €	20.175,00 €
Summe	20.658,79 €	18.333,79 €	21.163,79 €	29.826,04 €	89.982,41 €
19% MwSt.	3.925,17 €	3.483,42 €	4.021,12 €	5.666,95 €	17.096,66 €
Summe Gesamt	24.583,96 €	21.817,21 €	25.184,91 €	35.492,99 €	107.079,07 €

Abbildung 1: Angebotsvergleich Fa. Löther – Kartonagenerkennung

A.2 Auswertung IR-Sensor

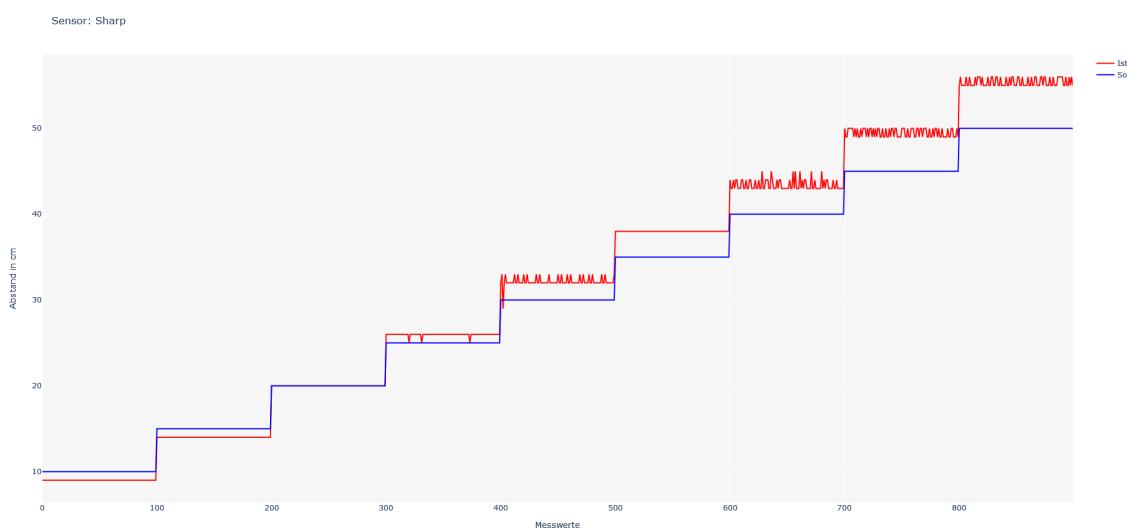


Abbildung 2: Auswertung IR-Sensor

A.3 Auswertung Ultraschallsensor

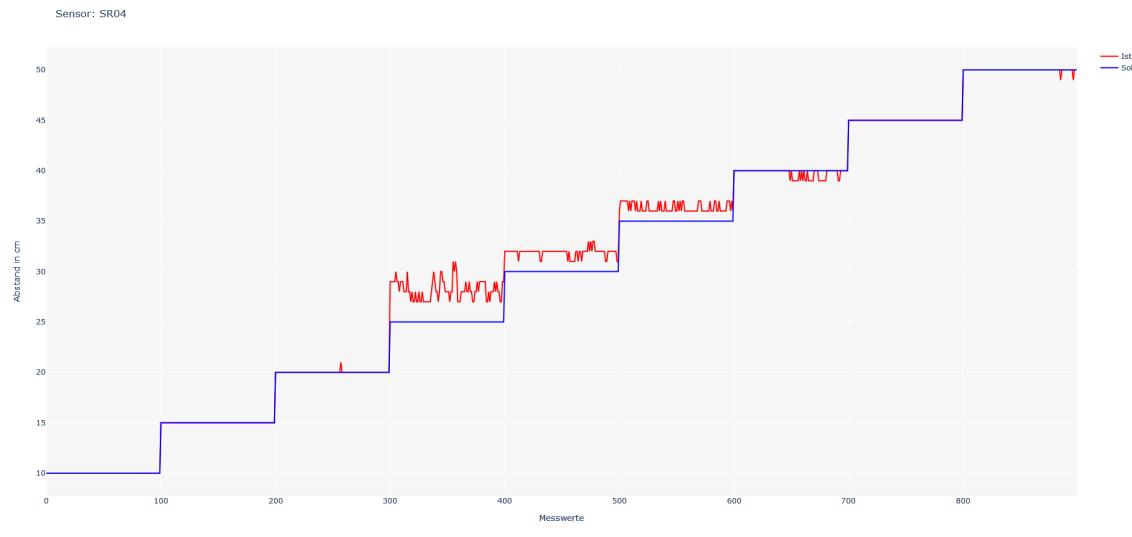


Abbildung 3: Auswertung Ultraschallsensor

A.4 Auswertung Lasersensor

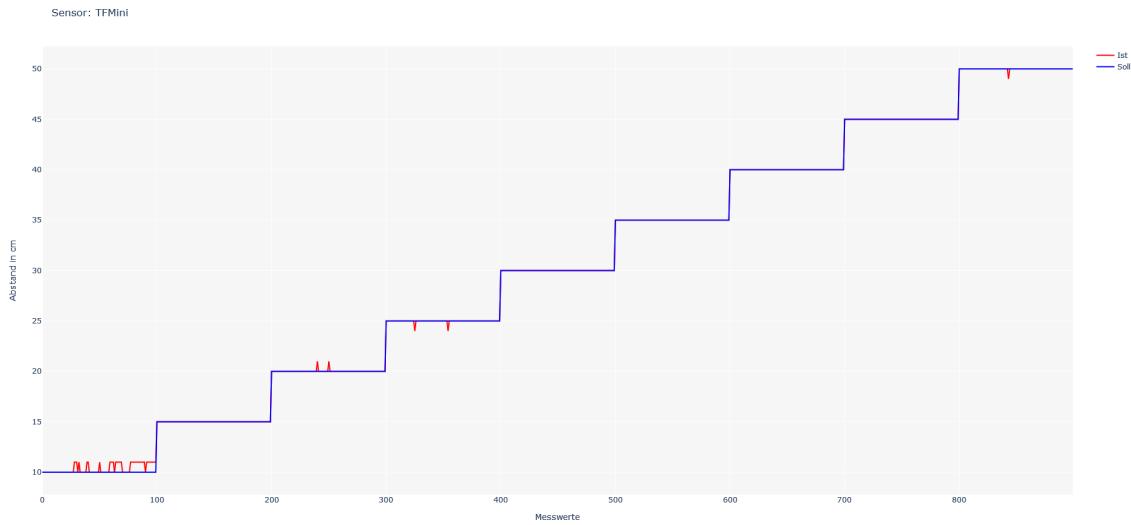


Abbildung 4: Auswertung Lasersensor

A.5 Tabellenmodell

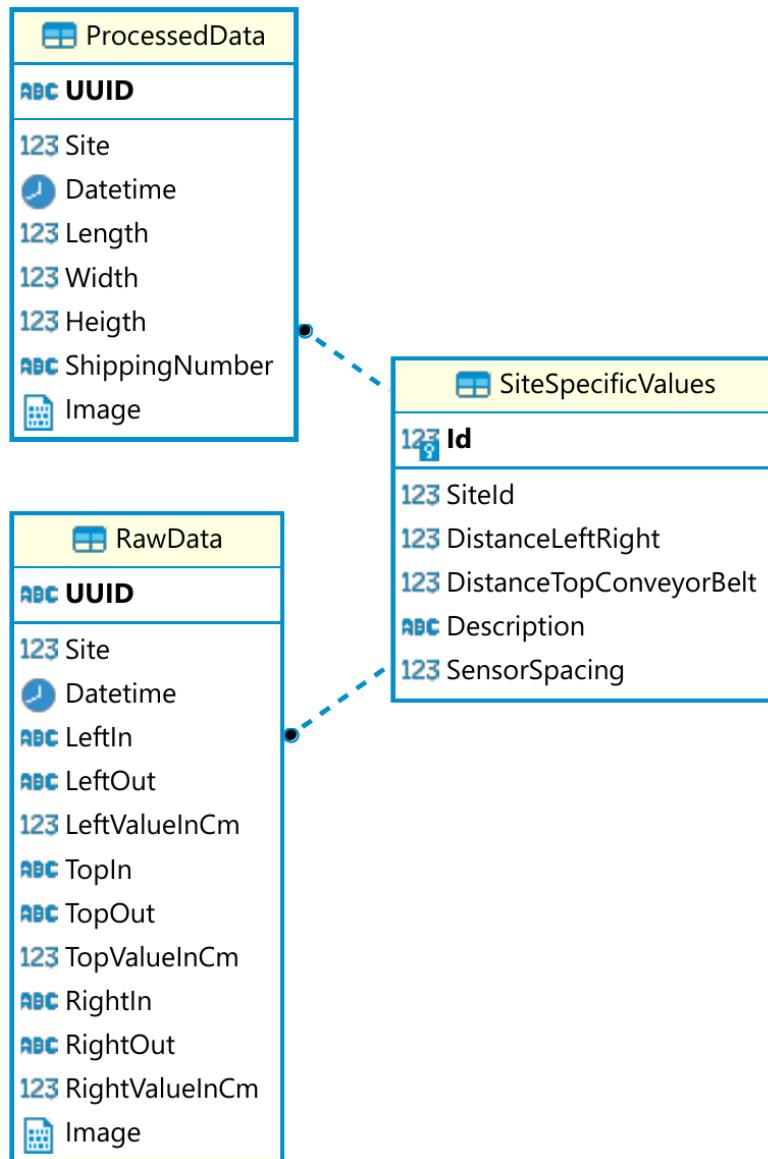


Abbildung 5: Tabellenmodell

A.6 Sensorhalterung

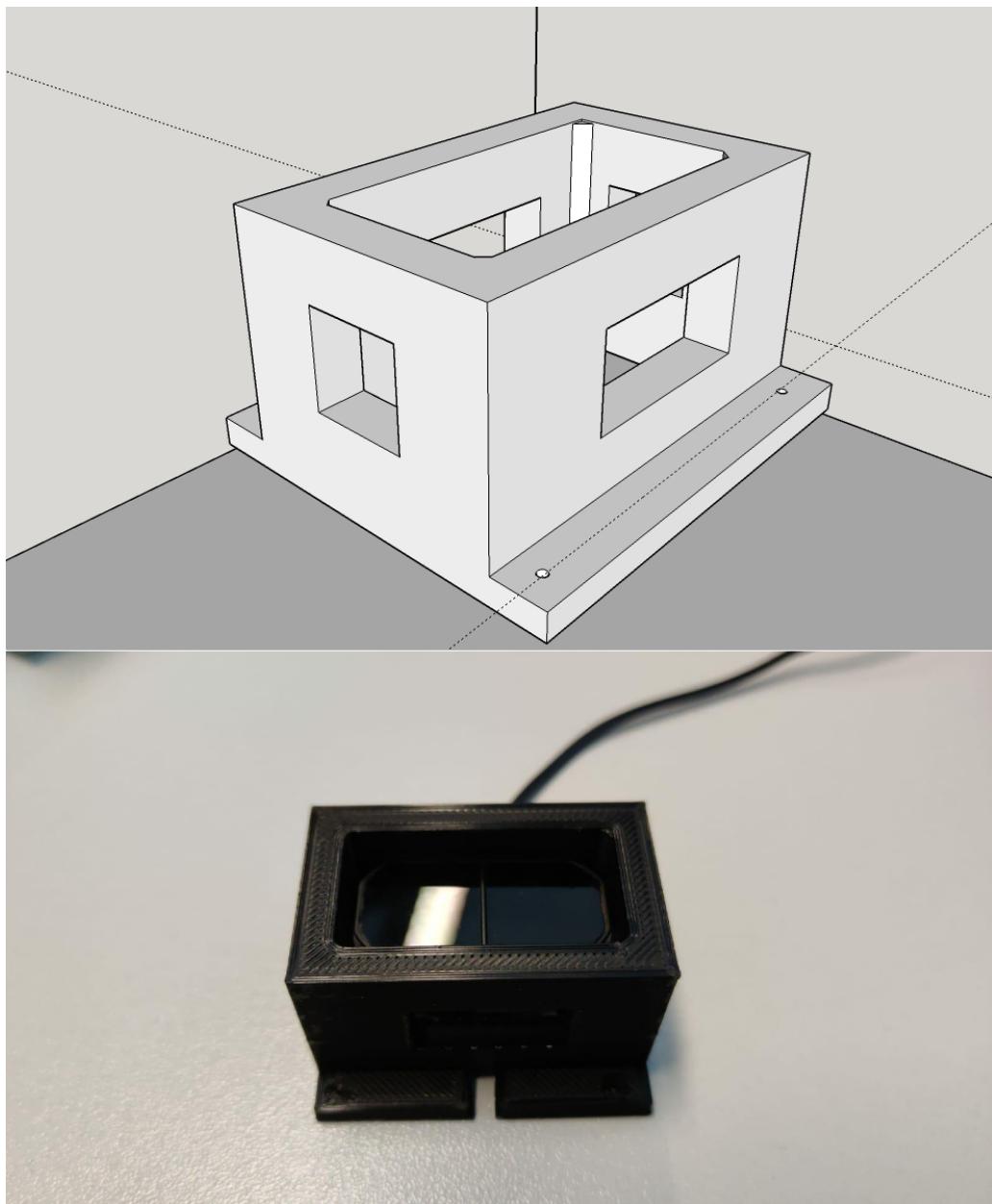


Abbildung 6: Sensorhalterung: Vorlage [3] und zusammengesetztes Endprodukt

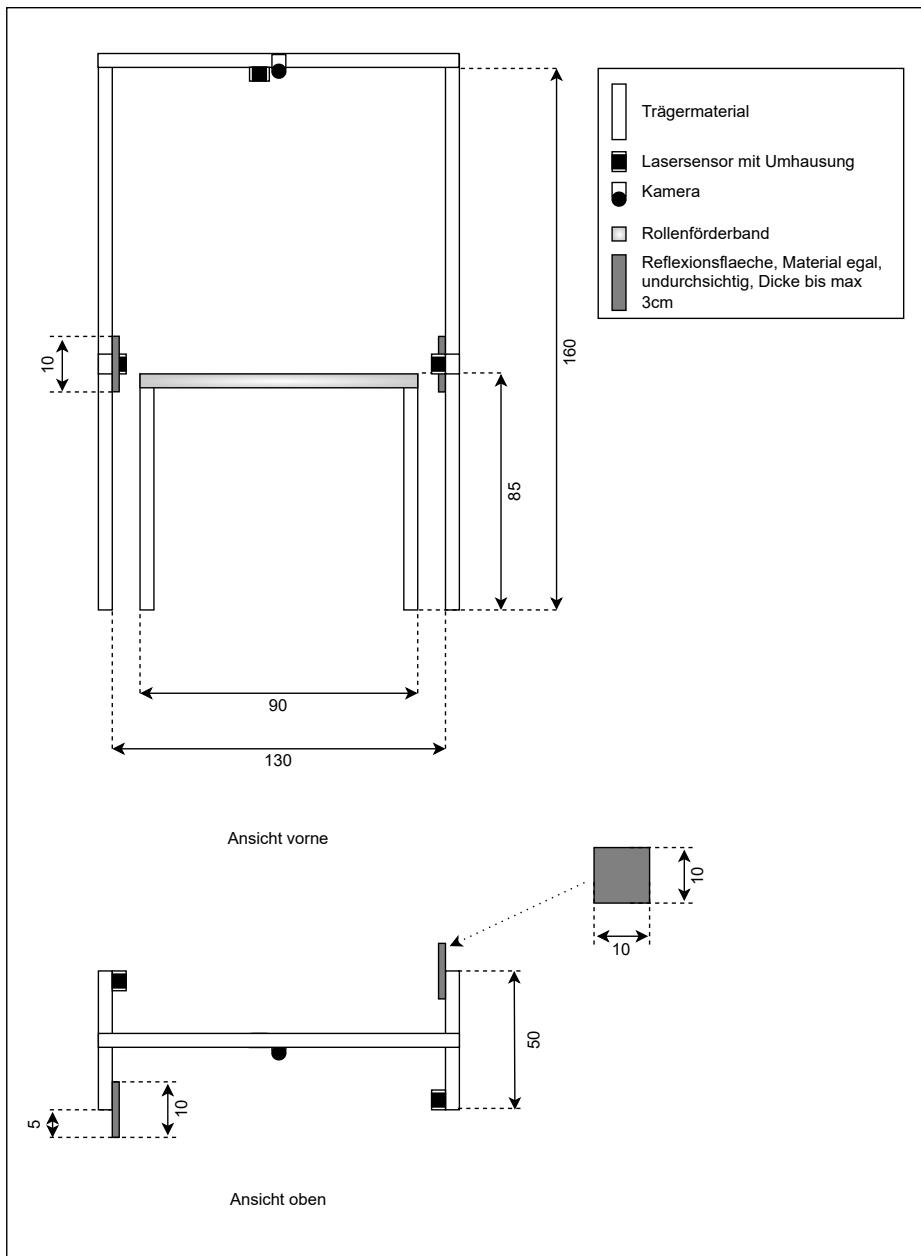
A.7 2D-Sensorhalterung

Abbildung 7: Sensorträger 2D

A.8 3D-Sensorträger

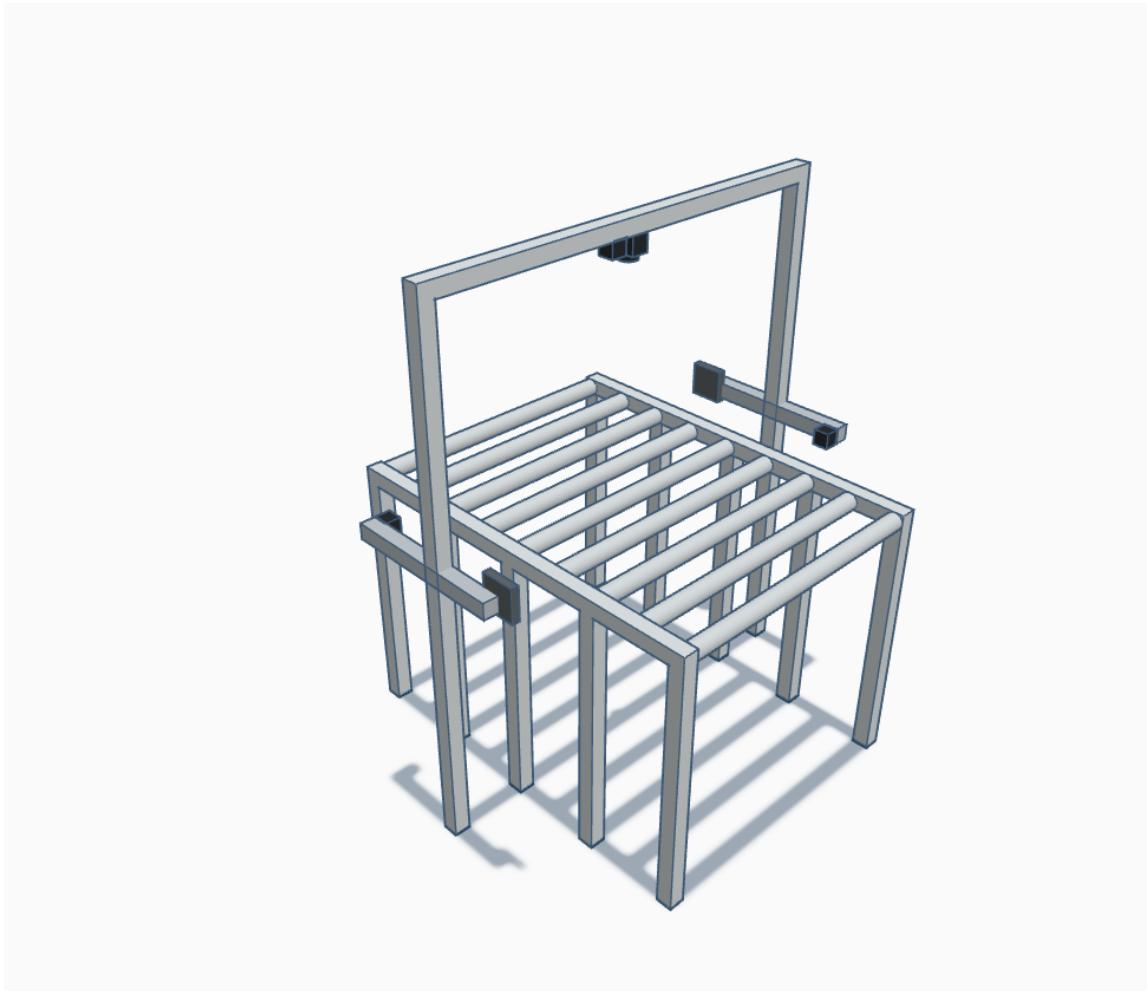


Abbildung 8: Sensorträger 3D gezeichnet

A.9 Fertiger Sensorträger



Abbildung 9: Fertig montierter und aufgestellter Sensorträger

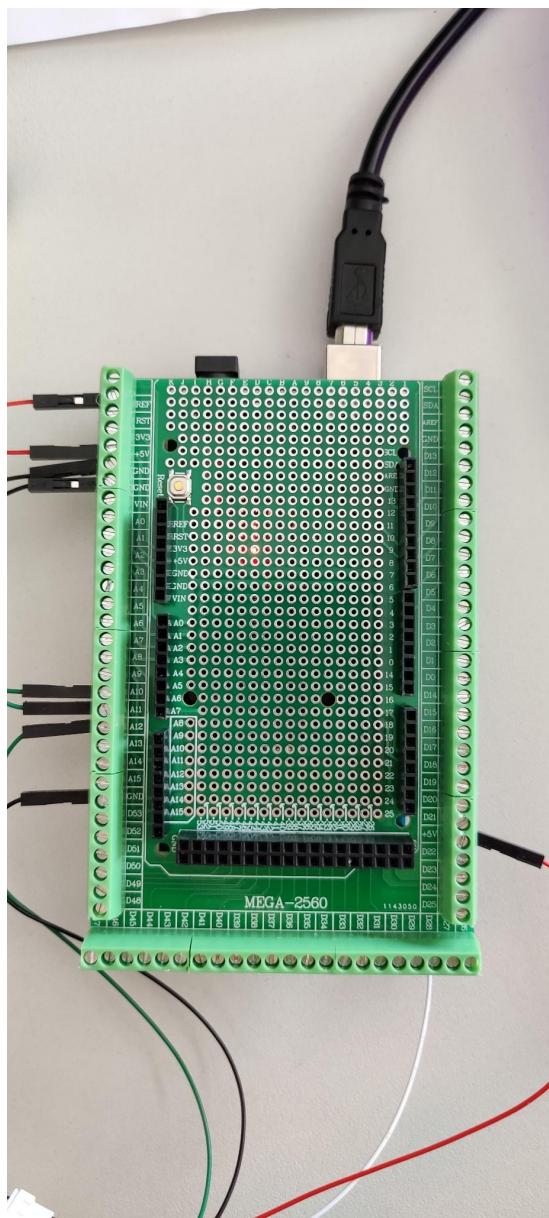
A.10 Arduino mit Lasersensoren

Abbildung 10: Arduino mit angeschlossenen Sensoren und Shield

A.11 Beispiel Verwendung labelImg

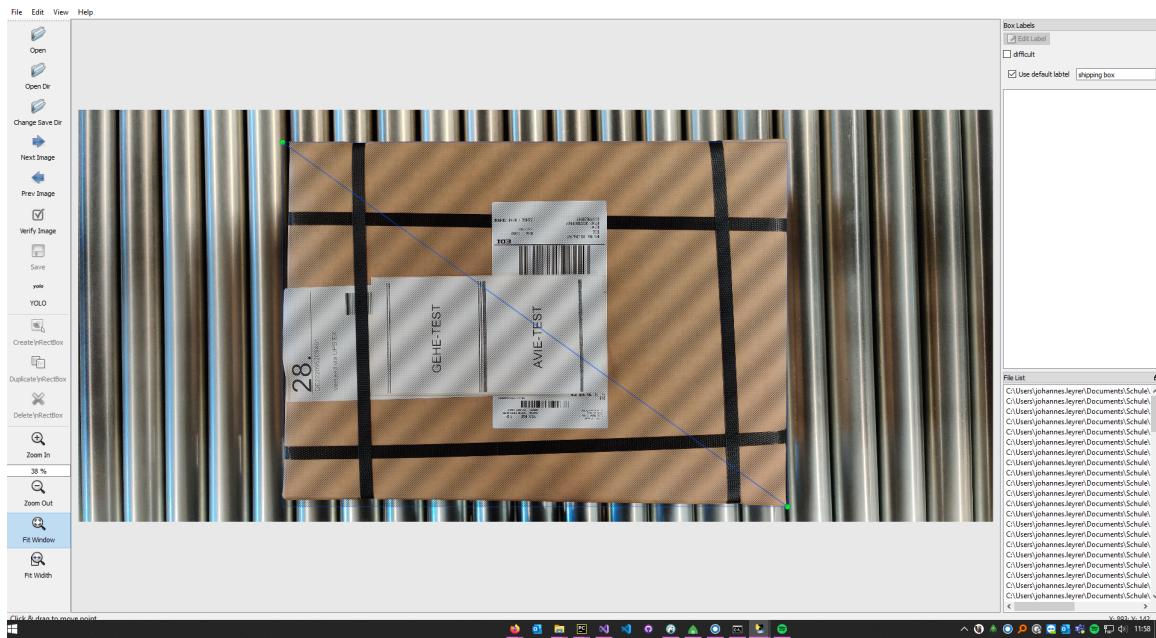


Abbildung 11: Beispiel Verwendung labelImg

A.12 Beispiel Barcodeerfassung

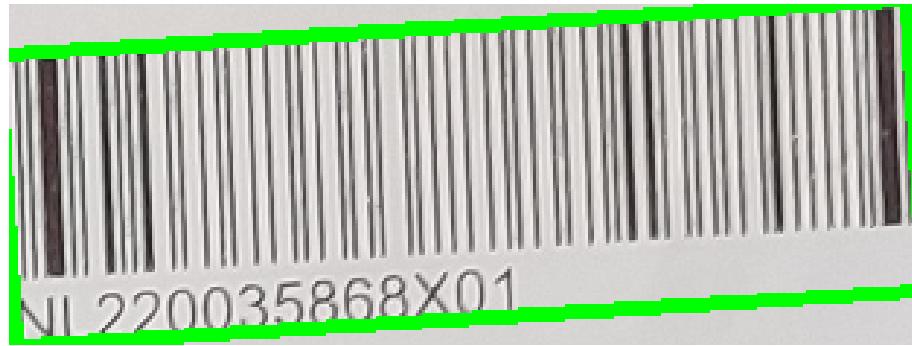


Abbildung 12: Beispiel Barcodeerfassung

A.13 Beispiel Versandverpackungserkennung mit YOLOv7

Abbildung 13: Beispiel Versandverpackungserkennung mit *YOLOv7*

A.14 Beispiel Labelerfassung



Abbildung 14: Beispiel Labelerfassung

A.15 GitLab-CI/CD

Listing 3: .gitlab-ci.yml-Datei für CI/CD

```
1   image: docker/compose:alpine-1.29.2
2
3   stages:
4     - build
5     - deploy
6
7   before_script:
8     - docker login -u $CI_DEPLOY_USER -p $CI_DEPLOY_PASSWORD
9       $CI_REGISTRY
10
11  build-prod:
12    stage: build
13    tags:
14      - faip-swarm-prod
15    script:
16      - docker build -f ./FAIP.PaMesAn.DataSaver/Dockerfile -t
17        ${CI_REGISTRY_IMAGE}/prod .
18      - docker push ${CI_REGISTRY_IMAGE}/prod
19    only:
20      refs:
21        - main
22
23  deploy-prod-raw:
24    stage: deploy
25    tags:
26      - faip-swarm-prod
27    script:
28      - docker stack deploy -c prod.raw.docker-compose.yml
29        --with-registry-auth pamesan
30    only:
31      refs:
32        - main
33
34  deploy-prod-processed:
35    stage: deploy
36    tags:
37      - faip-swarm-prod
38    script:
```

```
39      – docker stack deploy –c prod.processed.docker-compose.yml
40      --with-registry-auth pamesan
41  only:
42    refs:
43      – main
```

A.16 SQL-Erstellungsskript für RawData

Listing 4: SQL-Skript für RawData

```
1  CREATE TABLE PAMESAN.dbo.RawData (
2      UUID uniqueidentifier NOT NULL,
3      Site int NOT NULL,
4      [Datetime] datetime NULL,
5      LeftIn nvarchar(50) COLLATE Latin1_General_CI_AS NULL,
6      LeftOut nvarchar(50) COLLATE Latin1_General_CI_AS NULL,
7      LeftValueInCm int NULL,
8      TopIn nvarchar(50) COLLATE Latin1_General_CI_AS NULL,
9      TopOut nvarchar(50) COLLATE Latin1_General_CI_AS NULL,
10     TopValueInCm int NULL,
11     RightIn nvarchar(50) COLLATE Latin1_General_CI_AS NULL,
12     RightOut nvarchar(50) COLLATE Latin1_General_CI_AS NULL,
13     RightValueInCm int NULL,
14     [Image] varbinary(MAX) NULL,
15     CONSTRAINT PK_RawData PRIMARY KEY (UUID),
16     CONSTRAINT RawData_FK FOREIGN KEY (Site) REFERENCES
17     PAMESAN.dbo.SiteSpecificValues(Id)
18 );
```

FLYERALARM Industrial Print GmbH
Entwicklerdokumentation

Projekt PaMesAn

System zur Erfassung von Versandverpackungen mit Hilfe von
Bild- und Sensordaten zur Erfüllung der Novelle des
Verpackungsgesetzes

Ersteller
Johannes Leyrer
IT – Kesselsdorf

Historie:

2022.10.31: Erstellung

B Entwicklerdokumentation PaMesAn

Zweck der PaMesAn-Umgebung ist es, die Abmessungen der Kartonagen im Versand nach dem Aufdrucken des Shipping-Labels zu erfassen und ein Bild aufzunehmen. Diese Daten sollen mehreren Services bereitgestellt werden, bisher dem Datenspeicherservice und dem Datenverarbeitungsservice. Die Daten werden durch den Datenspeicherservice persistent in unbearbeiteter und bearbeiteter Form gespeichert. Der Datenverarbeitungsservice erkennt das Paket im Bild, das Shipping-Label und berechnet die Abmessungen des Pakets.

Alle verwendeten Pakete sind in den `requirements.txt` für Python-Projekte und in den Paket-abhängigkeiten bei C#-Projekten zu finden. Für die Programm, das auf dem Arduino ausgeführt wird, wurde `SoftwareSerial` verwendet.

B.1 Sensorträger und Komponenten

Der Sensorträger wurde anhand der in Abb. 15 auf Seite B-2 und Abb. 16 auf Seite B-3 zu sehenden Zeichnungen für den Standort Kesselsdorf von der Haustechnik KE angefertigt. Für Replikationen und Aufbau bitte an die Haustechnik KE wenden, ein Anschauungsobjekt steht in Kesselsdorf an Versandlinie 1. Der Sensorträger wurde mit folgenden Komponenten bestückt:

Benewake TF MINI PLUS IP65 zertifizierte Lasersensoren, drei Stück

AZDelivery Mega 2560 R3 Einplatinencomputer auf Arduino-Basis

ARCELI Shield Board Kit Aufsatz für den Arduino mit Schraubklemmen für die Pin-Anschlüsse

Dell Wyse 5070 Thin Client Windowsrechner ohne besondere Ausstattung

Microsoft Lifecam Studio Webcam, um ein Bild aufzunehmen (Anmerkung: In Zukunft möglicherweise auf KEYENCE SR-X100 AI-Codeleser wechseln)

Die Zeichnungen für den Sensorträger sind auf dem internen Speicher zu finden:

`\nas-mar-06\FAIP-Software\[NAME]\FAIP.PaMesAn\Sensortraeger\Zeichnungen`

Die Halterungen der Sensoren sind 3D-gedruckt, die Vorlage und Dateien sind im Internet oder auf dem internen Speicher zu finden:

<https://www.thingiverse.com/thing:4592503>

`\nas-mar-06\FAIP-Software\[NAME]\FAIP.PaMesAn\Sensortraeger\Sensoren`

Die Sensoren sind mit 4-adrig verschraubbaren Male-Sensorkabeln versehen, zu sehen in Abb. 17 auf Seite B-3. Die Anordnung ist bei Replikation unbedingt beizubehalten, um den Wartungsaufwand gering zu halten. Die ausgehenden Kabel sind wie in Abb. 18 auf Seite B-3 zu sehen anzuschließen. Sollten die Datenkabel an anderen A-Pins angeschlossen werden, muss dies im Auslesecode berücksichtigt werden. Ebenso sollten die Sensoren der Reihe von seitlich, oben

und seitlich nach an A9-A11 angeschlossen werden, wobei A9 mit dem Sensor verbunden ist, der zuerst einen Wert schickt, sollte ein Paket erfasst werden.

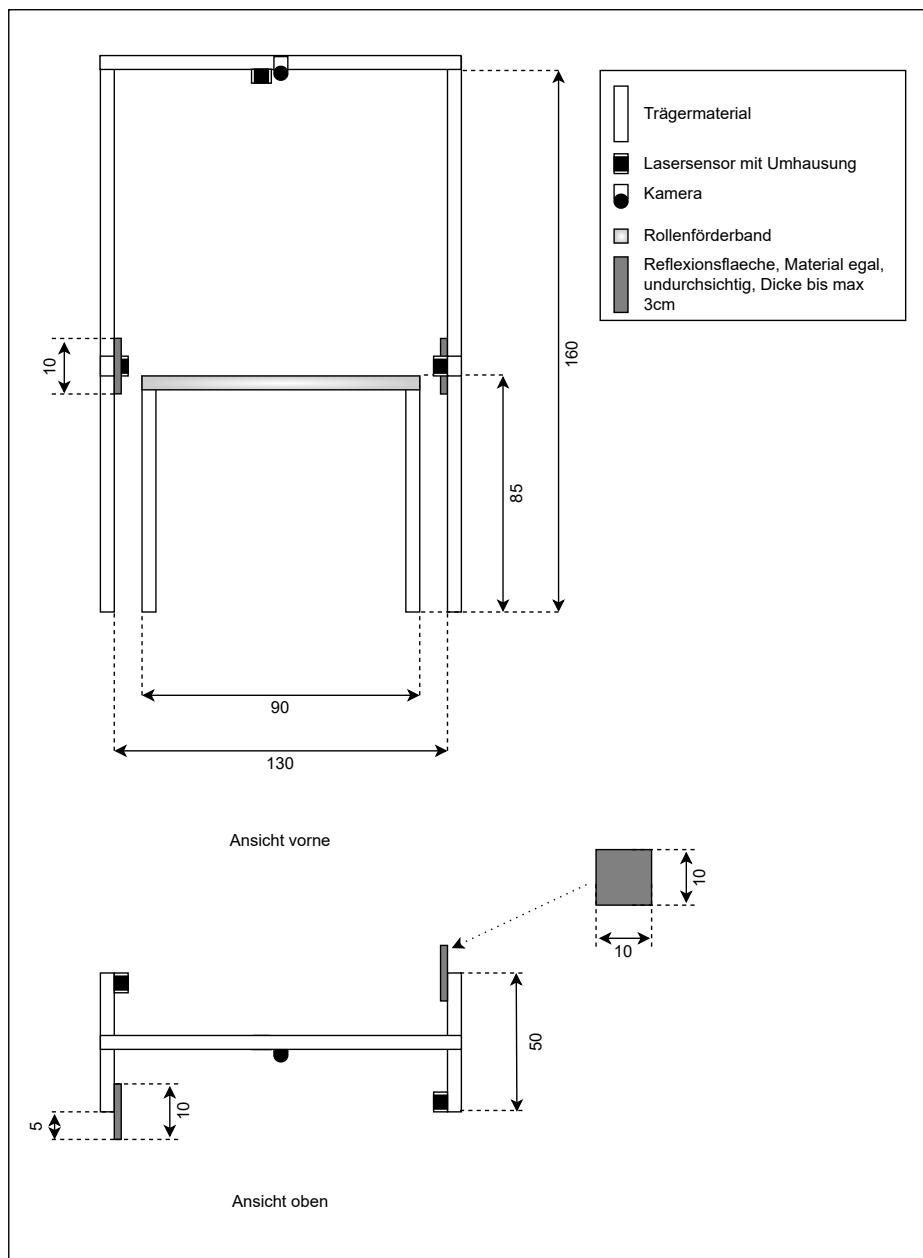


Abbildung 15: 2D-Zeichnung Sensorträger



Abbildung 16: 3D-Zeichnung Sensorträger

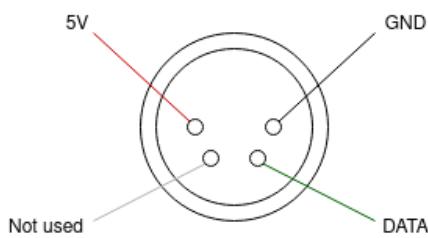


Abbildung 17: Anschlusspins der 4-adrig ver-schraubbaren Male-Sensorkabel

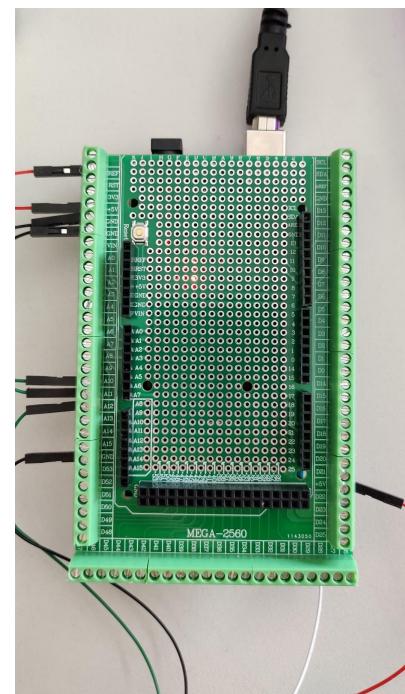


Abbildung 18: Pinbelegung am Arduino

B.2 Datenspeicherservice

Der Datenspeicherservice baut auf dem FAIP.LIB.RMQ-Paket auf. Dieses ist über die interne NuGet-Schnittstelle zu beziehen. Die Dokumentation zu diesem Paket ist im GitLab zu finden unter:

[https://gitlab.druckhaus-mainfranken.de/\[NAME\]/faip.lib.rmq](https://gitlab.druckhaus-mainfranken.de/[NAME]/faip.lib.rmq)

Der Service selbst ist unter folgender URL zu finden:

[https://gitlab.druckhaus-mainfranken.de/\[NAME\]/faip.pamesan.datasaver](https://gitlab.druckhaus-mainfranken.de/[NAME]/faip.pamesan.datasaver)

Programm.cs

Receiver_ConsumerReceivedAsync In der *Programm.cs* ist diese Methode zum Empfangen der Daten zuständig. Sie nimmt die *BasicDeliverEventArgs* entgegen, versucht diese zu einem UTF8-String zu parsen leitet diesen an die Speichermethode weiter. Etwaige Fehler werden nach FAIP.LIB.RMQ-Anforderung abgefangen und entsprechend gehandhabt.

internal static class SaveDataToDb

SaveData(string data) Diese Methode nimmt die empfangenen Daten entgegen, stellt die Verbindung zur Datenbank her und verteilt je nach RabbitMQ-Exchange-Name die Daten und den DbContext an die beiden folgenden Methoden weiter.

SaveDataToRaw(string data, PAMESANContext dbContext) Hier werden die JSON-Daten deserialisiert, auf das RawData-Modell geparsert und mit Hilfe von EFCore in die Tabelle RawData gespeichert.

SaveDataToProcessed(string data, PAMESANContext dbContext) Hier werden die JSON-Daten deserialisiert, auf das ProcessedData-Modell geparsert und mit Hilfe von EFCore in die Tabelle ProcessedData gespeichert.

B.3 Datenausleseservice

Der Datenausleseservice liest die Sensor- und Bilddaten aus und erkennt ein Paket anhand der Sensordaten. Die gemessenen Distanzwerte sowie das aufgenommene Bild werden mit RabbitMQ versendet. Der Service besteht aus einem C-Programm, das auf dem Arduino und einem Python-Programm, das auf dem Windowsrechner läuft. Das gemeinsame GitLab-Repository ist unter folgender URL zu finden:

[https://gitlab.druckhaus-mainfranken.de/\[NAME\]/faip.pamesan.datareader](https://gitlab.druckhaus-mainfranken.de/[NAME]/faip.pamesan.datareader)

Das Arduino-Programm initialisiert zunächst die Pins, an denen die Sensoren angeschlossen sind. Hier ist darauf zu achten, dass wie in Anhang B.1 auf Seite B-1 beschrieben, die Datenpins der Sensoren mit den initialisierenden Pins übereinstimmen. Standardmäßig sind das die Pins A9, A10 und A11. Danach gibt es die im folgenden näher erläuterten drei Methoden, die der Einrichtung, zum Auslesen eines Sensors und zur ständigen Wiederholung dessen pro Sensor dienen. Die `getTFminiData`-Methode wurde aus dem bereitgestellten Softwareprogramm des Herstellers Benewake verwendet, zu finden unter:

<https://github.com/TFmini/TFmini-Arduino>

setup() Diese Methode initialisiert die Baudaten des seriellen Schnittstellen zu den Sensoren und zum USB-Ausgang.

getTFminiData(SoftwareSerial* port, int* distance, int* strength, boolean* complete) Diese Methode liest die Daten des übergebenen seriellen Ports aus, analysiert die Daten, gibt bei Erfolg die Distanz und die Signalstärke zurück und setzt das `complete`-Flag auf True.

loop() Diese Methode bildet eine Schleife, die die einzelnen Sensoren abfragt und die Werte abgreift. Diese Werte werden dann als String über USB-Schnittstelle ausgegeben.

Das Python-Programm besteht aus vier Prozessen, die nach dem Auslesen der UmgebungsvARIABLEN und dem Initialisieren der seriellen Schnittstelle ausgeführt werden. Diese vier Prozesse und die Initialisierung werden durch vier Klassen repräsentiert, deren wichtigste Eigenschaften im folgenden kurz aufgelistet sind. Für die genaue Funktion einzelner Methoden bitte den Dokument-String der Methode beachten.

SerialInitializer Diese Klasse initialisiert die serielle Schnittstelle und stellt die Schwellwerte zu Verfügung.

_initialize_thresholds(ser: any) → tuple[int, int, int] Diese Methode initialisiert die Schwellwerte. Sie liest über einen Zeitraum von zehn Sekunden die Daten aus, wählt für jeden Sensor den höchsten Wert aus, zieht die den Durchschnitt der zehn höchsten gemessenen Werte von diesem Wert ab und bildet daraus den Schwellwert.

EnvConfig Diese Klasse liest die Umgebungsvariablen aus und stellt diese zur Verfügung.

DataReader Diese Klasse liest die Daten der seriellen Schnittstelle aus.

read_data(data_queue: Queue) Diese Methode liest die serielle Schnittstelle aus und parst diese zu einem Dictionary. Bei Erfolg sendet sie die Daten weiter über die `data_queue`.

evaluate_data(data_queue: Queue, evaluate_queue: Queue, thresh_1: int, thresh_2: int, thresh_3: int) Diese Methode empfängt die Daten über die `data_queue` und analysiert die Werte. Liegen diese unterhalb des Schwellwerts, werden die Daten in einer Liste gespeichert. Überschreiten alle Distanzwerte den jeweiligen Schwellwert wird die Liste an die `evaluate_queue` übergeben.

CamReader In dieser Klasse wird die Kamera initialisiert und eine Methode zum aufzunehmen eines Bildes bereitgestellt.

read_cam(evaluate_queue: Queue, api_queue: Queue) Diese Methode empfängt durch die `evaluate_queue` Daten. Bei einer 1 wird der aktuelle Frame zwischengespeichert. Bei einem Dictionary wird dieses mit dem zwischengespeicherten Bild angereichert und an die `api_queue` übergeben.

Publisher Diese Klasse stellt die Verbindung zu RabbitMQ her und versendet die von den Prozessen ermittelten Daten.

B.4 Datenverarbeitungsservice

Der Datenverarbeitungsservice erhält die durch die Sensoren gemessenen Distanzen und das dazugehörige Bild über RabbitMQ. Aus den Distanzen werden die Abmessungen des Pakets berechnet. Im Bild wird das Paket erkannt und zugeschnitten. In diesem Zuschnitt wird das Shipping-Label und der darauf liegende Barcode erkannt und ausgelesen. Diese Daten werden dann über RabbitMQ veröffentlicht.

RMQ Diese Klasse stellt die Verbindung zu RabbitMQ her, empfängt die rohen Sensor- und Bilddaten und versendet die berechneten Abmessungen, das ausgelesene Shipping-Label und das Bild mit hervorgehoben erkanntem Paket.

CardboardCalculator Diese Klasse berechnet Länge, Breite und Höhe des Pakets anhand der Sensordaten.

EnvConfig Diese Klasse liest die Umgebungsvariablen aus und stellt diese zur Verfügung.

CardboardDetector Diese Klasse initialisiert das YOLOv7-Modell und stellt die Erkennung des Pakets bereit.

init_model(weights: any) Diese Methode lädt die Gewichte für YOLOv7 und bereitet die Bilderkennung vor.

detect(img: numpy.ndarray) → tuple[int, int, int, int] Diese Methode erkennt das Paket im Bild und gibt die x-y-Koordinaten des Rechtecks sowie dessen Breite und Höhe zurück, das um das Paket gezeichnet wird.

trim_image(img: numpy.ndarray, x: int, y: int, width: int, height: int) → numpy.ndarray Diese Methode schneidet das Bild entsprechend der x-y-Koordinaten und der Breite und Höhe zu.

BarcodeDetector Diese Klasse stellt die Methoden zur Erkennung des Barcodes auf dem Paket bzw. auf dem Shipping-Label.

detect_barcode(img: numpy.ndarray) → tuple[int, int, int, int] Diese Methode erkennt den Barcode im Bild und gibt die x-y-Koordinaten des Rechtecks sowie dessen Breite und Höhe zurück, das um den Barcode gezeichnet wird.

trim_barcode(img: numpy.ndarray, x: int, y: int, width: int, height: int) → numpy.ndarray Diese Methode schneidet das Bild entsprechend der x-y-Koordinaten und der Breite und Höhe zu, sodass nur noch der Barcode im Bild ist.

BarcodeReader Diese Klasse stellt die Methode zum Auslesen des Barcodeerfassung

Die Bilderkennung wurde mit YOLOv7 umgesetzt, das dazugehörige GitHub-Repository ist hier zu finden:

<https://github.com/WongKinYiu/yolov7>

Hier ein kurzes Tutorial, wie ein neues Modell trainiert werden kann:

1. Erstellen eines Datensatzes. Ab 50 Bildern kann ein gutes Modell trainiert werden, es kommt allerdings auf die Qualität des Bildmaterials an.

2. Beziehen des Label-Programms labelImg, verfügbar unter:

<https://github.com/heartexlabs/labelImg>

3. labelImg installieren und ausführen,

4. Im linken Reiter von PascalVOC auf YOLO umschalten.

5. Bildordner mit labelImg aufrufen und Speicherpfad für die Labels festlegen

6. Label erzeugen. Bei nur einem Label die „Single Class Label“-Option unter View einschalten.

7. Bilder labeln.

8. YOLOv7 beziehen von:

<https://github.com/WongKinYiu/yolov7>

9. In `yolov7/data/train` 70% der gelabelten Bilder in einen Ordner mit dem Namen `images` und die entsprechenden Labels in einen Ordner `labels` ablegen. Den Rest mit gleicher Ordnerstruktur in den Pfad `yolov7/data/val`.

10. In `yolov7/data/` eine `custom_data.yaml` ähnlich zu der Vorlage unter Listing 5 auf Seite B-9 anlegen, wobei `train` und `val` die Pfadangaben für die Trainings- und Validierungsdaten sind. `nc` steht für die Anzahl der Labels und welche durch Komma getrennt in das `names`-Array geschrieben werden.

11. Nun kann das Modell mit Hilfe des in Listing 6 auf Seite B-9 zu sehenden Befehls trainiert werden. Kurz die wichtigsten Begriffe: Mittels `--device` kann eine von möglicherweise mehreren Grafikkarten ausgewählt werden. Die `--batch-size` gibt an, wie viele Daten verarbeiten werden, bevor das Modell aktualisiert wird. `--epochs` bestimmt, wie viele Durchläufe durch den Trainingsdatensatz durchgeführt werden sollen. `--img` bestimmt die Größe der Bilderskalierung, hier ist „640 640“ der Sweetspot, da sonst zu viel VRAM verwendet wird und das Training länger dauert. `--data` gibt an, wo die `custom_data.yaml`-Datei liegt. `--name` gibt den Namen des Ausgabeordners an.

12. Nach erfolgreichem Training können die Gewichte für das eigene Model in `yolov7/runs/train/[--name]/weights` entnommen werden.

Listing 5: custom_data.yaml

```
1     train: ./data/train
2     val: ./data/val
3     nc: 1
4     names: ["shipping box"]
5
```

Listing 6: Befehl zum Trainieren des Modells

```
1 python train.py --device 0 --batch-size 16 --epochs 100 --img 640 640 --
  data data/custom_data.yaml --hyp data/hyp.scratch.custom.yaml --cfg cfg/
  training/yolov7.yaml --weights yolov7.pt --name yolo7-custom
2
```