

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

<somewhere>

<somewhere>

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.

**11.08.2022 09:16:42 - Statusausdruck für:
Leyrer, Johannes (468322)
Fachinformatiker/-in Anwendungsentwicklung (701701000000)**

**Ausbildungsbetrieb / Praktikumsbetrieb
FLYERALARM Industrial Print GmbH**

Projektbezeichnung

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

Projektbeschreibung

Zielgruppe und Auftraggeber Der Auftraggeber ist die Geschäftsleitung und die Zielgruppe sind die Mitarbeiter der Produktion. Als Ansprechpartner bezüglich fachlicher Anforderungen an das Projekt dienen die unterschiedlichen Fachabteilungen. Ist-Zustand Der Ist-Zustand besteht aus einer Konsolen-Anwendung, welche 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. Das manuelle Pflegen der Verpackungsdaten ist nicht mehr praxistauglich, Soll-Konzept Um von den Erfahrungswerten der verpackenden Mitarbeitenden zu profitieren, sollen die Pakete nach dem Verpacken und kurz vor dem Versand gescannt und somit dem entsprechenden Auftrag zugeordnet 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. Projektdurchführung Zur Erfassung der Versandverpackungen werden Distanzsensoren und eine Kamera eingesetzt. Zur Speicherung der Daten wird eine Datenbank auf einem bestehenden Microsoft SQL-Server 2017 verwendet. Für die Datenbankanbindung wird die ASP .NET Core 6.0 Web API, die Programmiersprache C# und das Entity Framework Core zur Abbildung verwendet. Die API wird mit einer GitLab Pipeline als Docker Container erstellt und automatisiert im Docker Swarm verteilt. Mithilfe von Python und verschiedener Bildverarbeitungs-Bibliotheken werden die Sensor- und Bilddaten ausgelesen. Zur Entwicklung der REST-API und des Sensor- und Bildverarbeitungs-Programms werden die IDEs Visual Studio 2022 und PyCharm Community verwendet. Das kostenlose universale Datenbank-Tool DBeaver Community wird verwendet um die Datenbankeinträge einzusehen.

Projektumfeld

Die FLYERALARM Industrial Print GmbH ist mit mehr als 2000 Mitarbeitern an acht Standorten in Deutschland vertreten. Neben der Herstellung verschiedenster Druckerzeugnisse werden diese Produkte auch weiterverarbeitet, montiert, konfektioniert und für den Versand vorbereitet. Das Kerngeschäft der IT-Abteilung ist hierbei das Aufarbeiten der Druckdaten von FLYERALARM, um diese den Mitarbeitern in der Produktion, den Druckmaschinen aber auch der Geschäftsführung zur Verfügung zu stellen.

Projektphasen (einschl. Zeitplanung)

1. Analyse 10h - Ist-Analyse 2h - "Make or buy"-Analyse 2h - Analyse der benötigten Libraries 2h - Analyse der benötigten Schnittstellen von Fremdsystemen 2h - Soll-Konzept 2h 2. Entwurf 8h - Entwurf der Datenbankstruktur 2h - Entwurf der REST-API 2h - Entwurf der Datenabfrage und -verarbeitung der Sensoren 4h 3. Implementierung 39h - Erstellen der Datenbank 4h - Erstellen der REST-API 9h * Datenbankanbindung 2h * Routenimplementierung 4h * Vergleich gemessener Werte mit bekannten Verpackungen 2h * Schnittstellendokumentation mit Swagger 1h - Platzierung der Sensoren 1h - Erstellen des Sensor-Abfrage-Workers 9h * Abfragen der Sensoren 8h * REST-Posts ab API 1h - Erstellen des Bildverarbeitungs-Workers 8h * Aufnahme und Abspeichern des Bildes 1h * Erkennen des Kartons im Bild 7h * Erkennen wichtiger Informationen aus dem Bild 7h * REST-Posts an API 1h 4. Test 5h - Code-Review 2h - Schreibtischtests 3h 5. Einführung 6h - Inbetriebnahme der API und der Sensor-Worker 5h - Abnahme durch Teamleiter 1h 6. Dokumentation 12h - Erstellen der Entwicklerdokumentation 4h - Erstellen der Projektdokumentation 8h Insgesamt 80h

Dokumentation zur Projektarbeit (nicht selbstständig erstellte Dokumente sind zu kennzeichnen)

Die Beschreibung der REST-Schnittstellen wird während der Implementierungsphase automatisch mit dem Swagger-Framework erstellt. Die Dokumentation der Projektarbeit wird stichpunktartig während der einzelnen Phasen im Projekt vorgenommen. Die Ausformulierung und die Entwicklerdokumentation werden am Ende des Projekts vorgenommen.

**Bearbeitungsdauer von
03.10.2022**

**Bearbeitungsdauer bis
11.11.2022**

Präsentationsmittel:

Overheadprojektor



Flipchart

andere Präsentationsmittel (sind vom Prüfling mitzubringen)

- Laptop - Beamer

Themenbetreuer

Herr Marco Thiergart E-Mail: marco.thiergart@flyeralarm-industrialprint.com Telefon: +49 9391 50314902

Auflagen

Grund Ablehnung

Abschlussprüfung im Beruf Fachinformatiker/-in Anwendungsentwicklung

Winter 2022/23

Themenbestätigung

Azubi-Nr.: 468322

Name: Johannes Leyrer

Thema bestätigt	X
Mit Auflagen bestätigt	
Grund Ablehnung	

ABSCHLUSSPRÜFUNG IT-BERUFE: NACHWEIS FÜR DEN BETRIEBLICHEN AUFRAG

- IT-Systemelektroniker
- Kaufmann für Digitalisierungsmanagement
- Fachinformatiker (Anwendungsentwicklung)
- Fachinformatiker (Daten- und Prozessanalyse)

- Kaufmann für IT-System-Management
- Fachinformatiker (Systemintegration)
- Fachinformatiker (Digitale Vernetzung)

Name, Vorname:

Leyrer, Johannes

Prüflingsnummer:

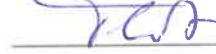
468322

Datum:	Zeitraum (Stunden):
05.10.2022	8
06.10.2022	8
12.10.2022	8
13.10.2022	8
19.10.2022	8
20.10.2022	8
24.10.2022	8
25.10.2022	8
26.10.2022	8
02.11.2022	8

Unterschrift
Prüfungsteilnehmer:



Unterschrift
Themenbetreuer:



Ich versichere, dass ich den betrieblichen Projektauftrag einschließlich Dokumentation selbstständig und nur mit den angegebenen Hilfsmitteln erstellt habe.

Ort, Datum:

Kesselsdorf, 07.11.2022

Unterschrift Prüfungsteilnehmer:



Ort, Datum:

Kesselsdorf, 07.11.2022

Unterschrift Themenbetreuer:



Datenschutz:

Die IHK Dresden ist für die Durchführung von Prüfungen in der Aus- und Weiterbildung zuständig. Die Ermächtigung zur Datenverarbeitung in diesem Zusammenhang ergibt sich aus Art. 6 Abs. 1 Buchstabe e DSGVO.

Hinweis: Für Prüfungsergebnisse und Unterlagen ergeben sich zum Teil vom Üblichen abweichende Aufbewahrungsfristen.

Prüfungsergebnisse aus der beruflichen Bildung werden 50 Jahre aufbewahrt, da über die Zeit des gesamten Erwerbslebens die Möglichkeit der Ausstellung einer Zeugniszweitschrift gewahrt werden muss. Prüfungsunterlagen werden hingegen ein Jahr nach Erlangen der rechtlichen Bestandkraft des Ergebnisses vernichtet.

Sie können Widerspruch gegen die Verarbeitung einlegen (Art. 21 DSGVO). Sollten Sie davon Gebrauch machen, prüft die IHK, ob die gesetzlichen Voraussetzungen hierfür erfüllt sind. Hinweis: Die zur Erfüllung der hoheitlichen Aufgaben notwendigen Daten können in der Regel nicht vor Ablauf der Speicherfrist gelöscht werden.

Die umfassende Datenschutzerklärung der IHK Dresden finden Sie unter <https://www.dresden.ihk.de/datenschutz>. Den Widerspruch können Sie durch Nutzung des [Widerspruchsformulars](#) auf der Webseite, schriftlich bei der IHK widerspruchds@dresden.ihk.de einlegen.

Ort, Datum:

Kesselsdorf, 07.11.2022

Unterschrift Prüfling:



Ort, Datum:

Kesselsdorf, 07.11.2022

Unterschrift Themenbetreuer:



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	4
3.1	Ist-Analyse	4
3.2	Soll-Analyse	4
3.3	„Make or buy“-Analyse	4
3.4	Analyse der benötigten Hardware	5
3.5	Analyse des Standorts	6
4	Entwurf	6
4.1	Zielplattform	6
4.2	Datenmodell	7
4.3	Architekturdesign	7
4.4	Entwurf des Sensorträgers	7
5	Implementierungsphase	8
5.1	Erstellung der Datenbanktabellen	8
5.2	Erstellen der RabbitMQ Exchanges	8
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	9
5.5	Erstellen der Sensordaten-Abfrage-Anwendung	10
5.5.1	Verwendete Pakete	10
5.5.2	Implementierung der Ausleselogik	10
5.6	Erstellen des Datenverarbeitungsservice	11
5.6.1	Verwendete Pakete	11
5.6.2	Zusammenspiel der einzelnen Komponenten	11
5.6.3	Erkennung der Versandverpackung	11
5.6.4	Auslesen des Versandlabels	12
5.6.5	Berechnung der Paketgröße	13

5.7	Bestückung des Sensorträgers	13
6	Abnahme- und Einführungsphase	13
6.1	Code-Review	13
6.2	Inbetriebnahme und Abnahme	14
7	Dokumentation	14
8	Fazit	14
8.1	Soll-/Ist-Vergleich	14
8.2	Lessons Learned	15
8.3	Ausblick	15
Glossar		IV
Abkürzungsverzeichnis		VI
Abbildungsverzeichnis		VII
Tabellenverzeichnis		VIII
Listings		IX
A	Anhang	A-1
A.1	Detaillierte Zeitplanung	A-1
A.2	Ressourcenplanung	A-2
A.3	Angebot der Elektro Löther GmbH	A-3
A.4	Auswertung IR-Sensor	A-3
A.5	Auswertung Ultraschallsensor	A-4
A.6	Auswertung Lasersensor	A-4
A.7	Epic	A-5
A.8	Architekturdesign	A-6
A.9	Tabellenmodell	A-7
A.10	Skizze der Versandanlage	A-8
A.11	Sensorhalterung	A-9
A.12	2D-Modell des Sensorträgers	A-10
A.13	3D-Modell des Sensorträgers	A-11
A.14	Fertiger Sensorträger	A-12
A.15	Arduino mit Lasersensoren	A-13
A.16	Beispiel Verwendung labelImg	A-14
A.17	Beispiel Barcode erfassung	A-14
A.18	Beispiel Versandverpackungserkennung mit YOLOv7	A-15
A.19	Beispiel Labelerfassung	A-16
A.20	GitLab-CI/CD	A-17
A.21	SQL-Erstellungsskript für RawData	A-18

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

1.1 Projektumfeld

Die FLYERALARM GmbH (FA)¹ 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 Mitarbeitende (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 erfassten Pakete und deren Versandlabel werden 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 so die Daten in Zukunft zuverlässiger erfasst und ausgewertet werden. Das automatische Erfassen reduziert einen großen Teil des Wartungsaufwandes 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. Weitere Vorteile sind die Einsparung von Kosten, da die genaue Menge der verwendeten Verpackungen bestimmt wird; die Qualitätskontrolle verbessert sich, weil eventuelle Schäden sofort nachvollzogen werden können; zusätzliche Daten über Verpackungsgrößen und Zielorte können erhoben werden, wodurch Kosten gespart werden können. Auch kann so eine bessere Qualitätskontrolle gewährleistet werden, da eventuelle Schäden besser nachvollzogen werden können. Ebenso kann ausgewertet werden, welche Verpackungsgrößen in welche Länder geliefert werden.

¹**Anmerkung:** 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“.

1.3 Projektschnittstellen

Die einzelnen Softwarekomponenten des Projekts kommunizieren mittels *RabbitMQ*, einer Open Source Message Broker Software, miteinander. Diese Software ist als *HA-Cluster* an jedem Standort bereits vorhanden.

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

Ein Entwurf des Sensorträgers wurde anhand der in Abschnitt 3.2 auf Seite 4 beschriebenen Anforderungen erstellt. Die endgültige Konstruktion, das Beschaffen der Materialien sowie der Aufbau wurde unter kontinuierlicher Rücksprache von den Haustechnikern des Standorts Kesseldorf 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 mit den bekannten Abmessungen der zur Verfügung stehenden Verpackungen und bekannten Bestellungen zu vergleichen. Auch das Aufsetzen und Anlegen der Datenbank und des Datenbankbenutzers sowie das Einrichten des Windowsrechners sind nicht Teil des Projekts. Auch der *RabbitMQ*-Service und *Docker Swarm* zählen zu den Ressourcen, die bereits vorhanden waren oder zur Verfügung gestellt wurden.

2 Projektplanung

2.1 Projektphasen

Für das Projekt stehen 80 Arbeitsstunden zur Verfügung. Diese werden auf verschiedene Projektphasen verteilt, zu sehen in Tabelle 1 auf der nächsten Seite. Eine detaillierte Zeitplanung ist im Anhang Tabelle 5 auf Seite A-1 zu finden.

2.2 Abweichungen vom Projektantrag

Im Projektantrag wurde im Unterpunkt „Projektdurchführung“ 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 abgespeichert werden. Kurz vor Projektstart wurde realisiert, dass

Tabelle 1: Grobe Zeitplanung

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

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 Tabelle 6 auf Seite A-2 entnommen werden. Auf die zusätzlich bezogene Hardware wird in Abschnitt 3.4 auf Seite 5 eingegangen.

2.4 Entwicklungsprozess

Im Projekt wird nach Scrum gearbeitet, einem agilen Entwicklungsprozess, welcher den Standardprozess innerhalb des IT-Teams von FA darstellt. So kann auf die sich ändernden Anforderungen eingegangen werden, die sich durch eine kontinuierliche Rücksprache mit Teamleiter, Schichtleiter der Versandanlage und der Haustechniker oder durch die verwendete Hardware und deren Aufbau ergeben. Ein Beispiel des Epics ist in Abb. 5 auf Seite A-5 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 viele verschiedene 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 Informationen werden in MSSQL-Datenbanken gespeichert und müssen regelmäßig manuell gepflegt werden. Zudem muss auf Sonderfälle für verschiedene Produkte geachtet 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 soll automatisch auf der Versandstraße erkannt, dessen Abmessungen erfasst und das Versandlabel erkannt werden. Diese Daten sollen mit einem Bild des Pakets persistent gespeichert werden. Die 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 A-3 zu sehende Angebot erstellt. In diesem rechnet die Firma Löther mit etwa 25 000 € pro Standort für Kameralhardware, 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 auf der nächsten Seite 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, wofür ein Pauschalbetrag von 15 € pro Stunde berechnet wird.

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 Summe der Gesamtkosten aus Tabelle 2 auf der nächsten Seite und Tabelle 3 auf der nächsten Seite ergibt bei einer Eigenentwicklung 3095,89 €, was etwa ein Zehntel der von der

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 €

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 die Eigenentwicklung präferiert.

3.4 Analyse der benötigten Hardware

Um die Abmessungen der Verpackungen 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 A-3 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 A-4 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 A-4 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 ist mit einer Auflösung von 1080p für die ersten Tests ausreichend. 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 Codelesers 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 Abb. 8 auf Seite A-8 zu sehen. Auf dieser Skizze ist erkennbar, dass es nur einen kleinen Bereich gibt, hier in grün markiert, der als Standort für den Sensorträger in Frage kommt.

4 Entwurf

4.1 Zielplattform

Das Projekt besteht aus vier *Microservices*, welche in den Programmiersprachen C, C# und Python umgesetzt wurden. 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. 7 auf Seite A-7 zu sehen.

4.3 Architekturdesign

Als Anwendungsarchitektur wurde die *Microservices-Architektur* gewählt. Anstatt die Software als *Monolith* aufzubauen, wird bei einer *Microservices-Architektur* der modulare Ansatz verfolgt. Dadurch können die einzelnen *Microservices* unabhängig voneinander deployed, gewartet und verwaltet werden. Von der losen Kopplung, dem separaten Deployment und der hohe Skalierbarkeit kann vor allem die ressourcenintensive Bilderkennung profitieren. In Abb. 6 auf Seite A-6 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 sowie 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 der vorherigen Seite 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. 10 auf Seite A-10, der 3D-Entwurf ist in Abb. 11 auf Seite A-11 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 Aufnahmen für M2-Stellschrauben als Befestigungsoption. Um etwas mehr Schutz vor Stoßschäden und eine bessere Option der Befestigung zu haben, wurden die Sensorhalterungen 3D gedruckt. Dazu wurde eine Vorlage von Thingiverse [[thingiverse](#)] verwendet. Die Vorlage und das zusammengesetzte Endprodukt ist in Abb. 9 auf Seite A-9 zu sehen.

5 Implementierungsphase

5.1 Erstellung der Datenbanktabellen

Nach der Analyse wurde das in Abb. 7 auf Seite A-7 zu sehende Tabellenmodell erstellt und auf Basis dessen die daraus resultierenden Tabellen. Als Beispiel für die Erstellung einer Tabelle ist das passende SQL-Skript für die RawData-Tabelle in Listing 4 auf Seite A-18 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. Die Entscheidung fiel jeweils auf einen Fanout-Exchange, einen für die unverarbeiteten und einen für die verarbeiteten Daten. Ein Fanout-Exchange verteilt die eingegangenen Nachrichten auf alle ihm bekannten Queues. Somit kann auch zukünftig sichergestellt werden, dass Daten von anderen Services abgerufen werden können. 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. Zudem wurde für das Loggen von wichtigen Informationen und Fehlermeldungen das NuGet-Paket NLog installiert.

5.3.2 Datenbankanbindung

Die Datenbankanbindung erfolgte mit Hilfe der `Microsoft.EntityFrameworkCore.Tools`. Diese bieten 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. 11 auf Seite A-11 und als fertiger Sensorträger in Abb. 12 auf Seite A-12 zu sehen. Bekannt ist der Abstand zwischen Förderband und dem nach unten messenden Sensors. Aus der Differenz zwischen diesem und dem gemessenen Abstand zwischen Paket und Sensor wird die Höhe bestimmt. Ebenso bekannt ist der Abstand zwischen linkem und rechtem Sensor. Aus der Differenz zwischen diesem und der Summe der beiden gemessenen Abstände zwischen Paket und Sensoren wird die Breite 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, $s1$ und $s2$, sind mit einem bekannten Abstand a voneinander versetzt angebracht. Es wird jeweils der erste und letzte Zeitpunkt, $t1$ und $t2$, erfasst, an dem ein Sensor einen Abstandswert gemessen hat. Aus dem bekannten Abstand zwischen linkem und rechtem Sensor a und der Differenz der Erfassungszeitpunkte Zeit kann nun die Bandgeschwindigkeit v_B , siehe Gleichung (1), 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 oberhalb des Förderbands um 10 cm nach hinten versetzt montiert, um ein vollständiges und gerades Bild vom Paket zu erhalten.

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 ermöglicht. Neben

den eingebauten Python-Modulen wurde zum Erfassen der Sensor- und Bilddaten opencv-python, die Implementierung von *OpenCV* in Python und pyserial verwendet. Zudem wurde das Paket pika verwendet, um die Kommunikation mit *RabbitMQ* zu ermöglichen.

5.5.2 Implementierung der Ausleselogik der Sensor- und Bilddaten

Die drei Lasersensoren sind mit einem Arduino verbunden, zu sehen in Abb. 13 auf Seite A-13. Für das Auslesen von mehreren Benewake TF MINI PLUS-Sensoren stellt der Hersteller ein Skript zur Verfügung, das für zwei Sensoren ausgelegt ist. [**tfmini-arduino**] 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 der jeweilige Schwellwert ermittelt. Danach ist das Auslesen, die Datenverarbeitung und die Kommunikation mit *RabbitMQ* in je einen dedizierten Prozess ausgelagert. Diese Prozesse sind mit Queues 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 zwischenzuspeichern. 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 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. [labelimg]

5.6.2 Zusammenspiel der einzelnen Komponenten

Der Kommunikationsdienst arbeitet als Consumer die eingegangenen Nachrichten der Queue des RawData-Exchanges ab. Diese Daten werden dann an den Erkennungsprozess für Versandverpackungen weitergeleitet.

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. [yolov7] 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. [wuttke_transfer]

Für das eigene Modell wurden 100 Fotos von Versandverpackungen aufgenommen und mit labelImg gelabelt. Ein Beispiel für den Label-Prozess ist in Abb. 14 auf Seite A-14 zu sehen. 70 der Bilder und Label wurden als Trainingsdaten verwendet, wobei die restlichen 30 Bilder als Daten

zur Validierung dienten. 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 für die Ausführung ist in Abb. 16 auf Seite A-15 zu sehen.

Mit dem Aufruf der Detection-Klasse werden die Gewichte geladen, das Modell vorbereitet sowie alle benötigten 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 mit diversen Filtern bearbeitet. 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. 17 auf Seite A-16. Sind mehrere Rechtecke vorhanden, wird das mit der größten Kontur ausgewählt. Anhand der Koordinaten des Rechtecks wird ein Bildausschnitt erstellt, zu sehen in Abb. 15 auf Seite A-14, 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 Methode, die die Verbindung zur Datenbank herstellt.

Die Berechnung erfolgt wie in Abschnitt 5.4 auf Seite 9 beschrieben. Je eine Methode ist für die Berechnung für Länge, Breite und Höhe zuständig.

5.7 Bestückung des Sensorträgers

Nachdem die Haustechnik den Sensorträger ähnlich zu der in Abb. 10 auf Seite A-10 zu sehenden Skizze gebaut hatte, musste dieser mit Sensoren, Kamera, Arduino und Windowsrechner bestückt sowie alle Kabel ordnungsgemäß 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 dem 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 Meinung und Verbesserungsvorschläge 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 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 A-17 zu sehen. Dadurch können zwei Services aus einer Anwendung heraus gestartet werden. Zudem 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 das Hinzufügen von byte-arrays zu JSON-Objekten unzulässig ist. 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 Sensorauslesesoftware 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 eines Tages 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 darauf 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. Auf das Trainieren eines neuen Modells oder den Aufbau des Sensorträgers wird dafür 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 und vertrauliche 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. Keine Probleme wurden durch das Umschwenken auf eine *Microservices-Architektur* kurz vor Projektbeginn verursacht. Während der Planung, Analyse und Umsetzung traten bis auf das Problem mit der Bilderkennung keine größeren oder unerwarteten Probleme auf. Dieses konnte jedoch durch den Einsatz von *YOLOv7* gelöst werden, was allerdings einen verschobenen Zeitplan zur Folge hatte. Deshalb musste der Schreibtischtest gestrichen und die Abnahme verkürzt werden, um die Vorgabe von 80 Stunden für die Projektrealisierung zu erfüllen. 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 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 beschrieben.

8.2 Lessons Learned

Im Laufe des Projektes sind drei große Punkte aufgefallen, welche für die Zukunft 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 praktisch und leicht einsetzbar erwiesen. 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. Zudem soll der Abteilung für Qualitätsmanagement der Zugriff auf die Daten ermöglicht werden, was durch eine Webanwendung gelöst werden soll. Das sind allerdings Erweiterungen und Anpassungen für zukünftige Projekte.

Glossar

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

Docker Swarm, ein Cluster aus Docker-Containern. 2, 7, 14

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

HA-Cluster, Hochverfügbarkeitscluster, eine Anzahl von vernetzten Computern/Servern, die mit einem Minimum an Ausfallzeiten zuverlässig genutzt werden können. [cluster]. 2

Jittern, bezeichnet das zeitliche Zittern in der Übertragung von Signalen. [jittern]. 5

Microservices, sind kleine, auf eine Reihe von Funktionen ausgelegte Services, die sich auf die Lösung eines bestimmten Problems konzentrieren. [aws_microservice]. 6, 7

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

Monolith, bezeichnet eine Softwarearchitektur, bei der eine Anwendung als eine einzelne und zusammenhängende Einheit gebaut wird. [monolith]. 7

OpenCV, stellt als Programmbibliothek Algorithmen für die Bildverarbeitung und Computer Vision zur Verfügung. [opencv]. 7, 10

RabbitMQ, eine Open Source Message Broker Software, die das Advanced Message Queuing Protocol (AMQP) implementiert. [rabbitmq]. 2, 7, 8, 10, 11, 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. [semantische_segmentierung]. 11

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

Verpackungsgesetz, setzt seit 2019 die europäische Verpackungsrichtlinie 94/62/EG in Deutschland um und regelt das Inverkehrbringen von Verpackungen sowie die Rücknahme und Verwertung von Verpackungsabfällen. [verpackungsgesetz]. 1

YOLOv7, auch „you only look once, version 7“, stellt als Programmzbibliothek einen hochmodernen Objektdetektor zur Verfügung. [yolov7]. 7, 11, 12, 14, 15

Abkürzungsverzeichnis

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

Abbildungsverzeichnis

1	Angebotsvergleich Fa. Löther – Kartonagenerkennung	A-3
2	Auswertung IR-Sensor	A-3
3	Auswertung Ultraschallsensor	A-4
4	Auswertung Lasersensor	A-4
5	Beispiel Epic	A-5
6	Skizze des Architekturdesigns	A-6
7	Tabellenmodell	A-7
8	Skizze der Versandanlage	A-8
9	Sensorhalterung: Vorlage [thingiverse] und zusammengesetztes Endprodukt . . .	A-9
10	2D-Modell des Sensorträger	A-10
11	3D-Modell des Sensorträgers	A-11
12	Fertig montierter und aufgestellter Sensorträger	A-12
13	Arduino mit angeschlossenen Sensoren und Shield	A-13
14	Beispiel Verwendung labelImg	A-14
15	Beispiel Barcodeerfassung	A-14
16	Beispiel Versandverpackungserkennung mit Test-Label	A-15
17	Beispiel Erfassung des Versandlabels mit Test-Label	A-16
18	2D-Zeichnung Sensorträger	B-2
19	3D-Zeichnung Sensorträger	B-3
20	Anschlusspins der 4-adrig verschraubbaren Male-Sensorkabel	B-3
21	Pinbelegung am Arduino	B-3

Tabellenverzeichnis

1	Grobe Zeitplanung	3
2	Kostenverteilung Hardware	5
3	Kostenverteilung Personal	5
4	Abschließender Zeitablauf	15
5	Detaillierte Zeitplanung	A-1
6	Ressourcenplanung	A-2

Listings

1	Portierungsbefehl der Datenbanktabellen zu C#-Modellen	9
2	Befehl zum Trainieren des Modells	12
3	.gitlab-ci.yml-Datei für CI/CD	A-17
4	SQL-Skript für RawData	A-18
5	custom_data.yaml	B-9
6	Befehl zum Trainieren des Modells	B-9

A Anhang

A.1 Detaillierte Zeitplanung

Tabelle 5: Detaillierte Zeitplanung

Analyse	10 h
1 Ist-Analyse	2 h
2 Soll-Analyse	2 h
3 „Make or buy“-Analyse	3 h
4 Analyse der benötigten Hardware	2 h
5 Analyse des Standorts	1 h
Entwurf	8 h
1 Datenbankentwurf	1 h
2 Funktionen und Kommunikation der Services	4 h
3 Entwurf des Sensorträgers	3 h
Implementierung	39 h
1 Erstellung der Datenbanktabellen	2 h
2 Erstellen der RabbitMQ Exchanges	1 h
3 Erstellen des Datenspeicherservices	4 h
4 Sensorträger skizzieren und planen	3 h
5 Erstellen der Sensordaten-Abfrage-Anwendung	8 h
5.1 Arduino-Logik	2 h
5.2 Zusammenspiel Lasersensoren und Webcam	5 h
5.3 Anbindung an RabbitMQ	1 h
6 Erstellen des Datenverarbeitungsservice	18 h
6.1 Erkennen der Versandverpackung	8 h
6.2 Auslesen des Versandlabels	6 h
6.3 Berechnung Paketgröße	3 h
6.4 Anbindung an RabbitMQ	1 h
7 Aufbau Sensorträger	3 h
Test	5 h
1 Code Review	3 h
2 Schreibtischtest	2 h

Fortsetzung auf nächster Seite

Tabelle 5: Detaillierte Zeitplanung (Fortsetzung)

Einführung	6 h
1 Inbetriebnahme der Services	4 h
2 Abnahme durch den Teamleiter	2 h
Dokumentation	12 h
1 Erstellung der Entwicklerdokumentation	4 h
2 Erstellung der Projektdokumentation	8 h
Gesamt	80 h

A.2 Ressourcenplanung

Tabelle 6: Ressourcenplanung

Personal	Hardware	Software
Auszubildender	HP Z2 Mini G4 Workstation	Windows 10
Teammitglieder	Server	Visual Studio 2022
Teamleiter & Projektbetreuer	Custom Desktop-PC	Visual Studio Code
Product Owner	Arduino Mega 2560	PyCharm CE
Techniker		Arduino IDE
Schichtleiter Versand		DBeaver
		Docker Desktop

A.3 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 €
Elektrik / 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.4 Auswertung IR-Sensor

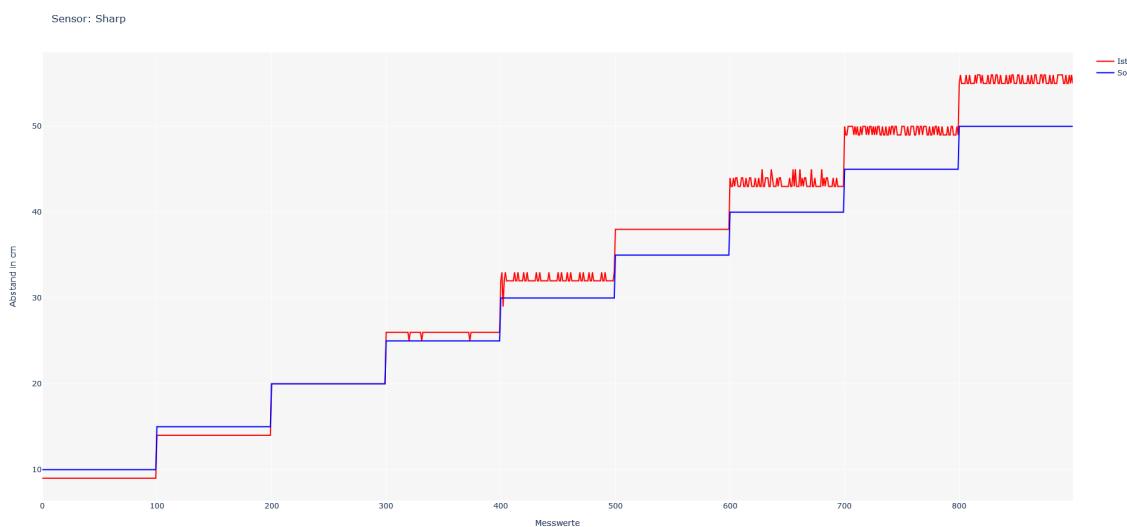


Abbildung 2: Auswertung IR-Sensor

A.5 Auswertung Ultraschallsensor

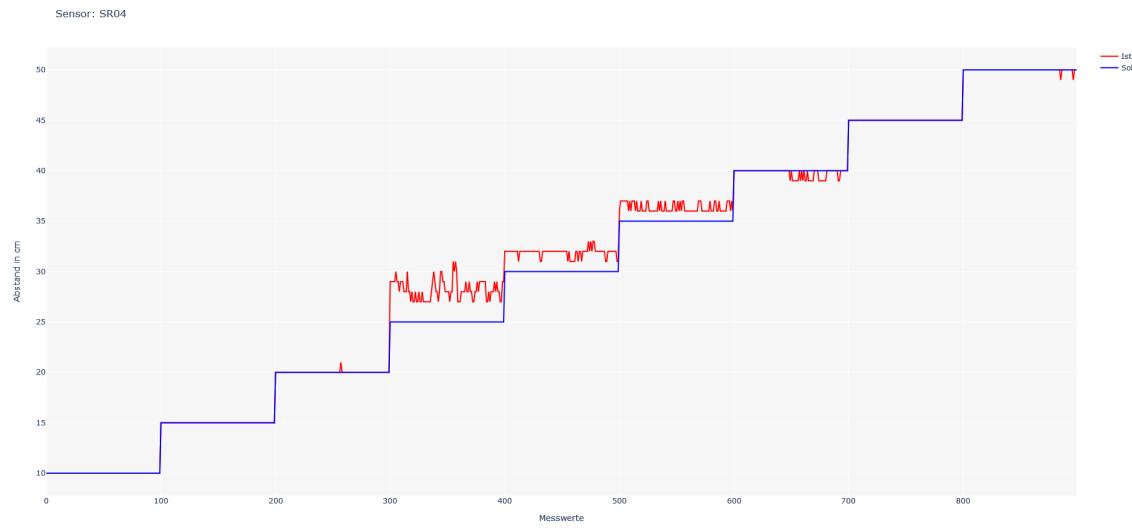


Abbildung 3: Auswertung Ultraschallsensor

A.6 Auswertung Lasersensor

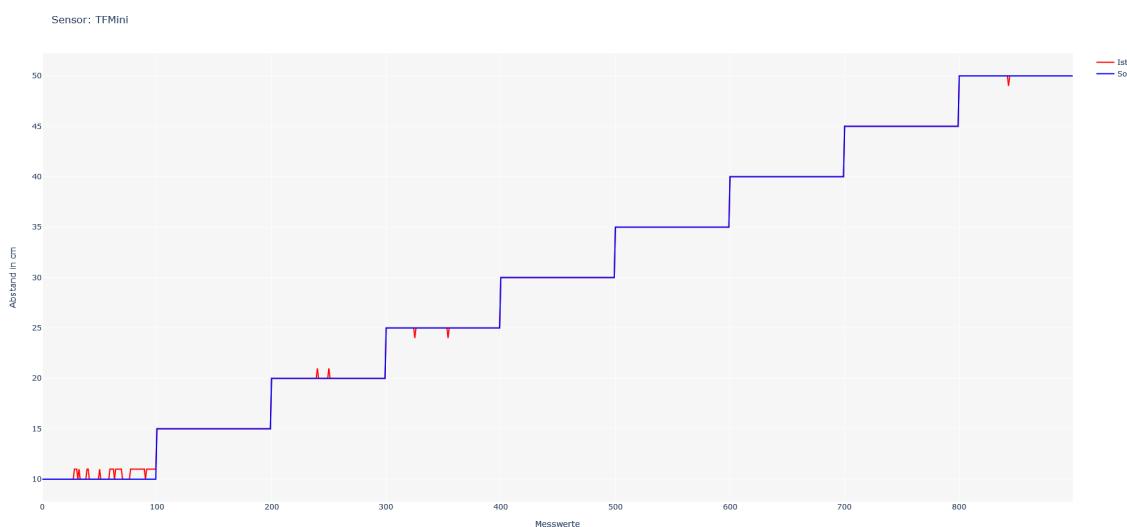


Abbildung 4: Auswertung Lasersensor

A.7 Epic

The screenshot shows a Jira Epic card for the project 'FAIP Softwareentwicklung / FAIPSE-3448'. The card title is 'Aufbau Produktverpackungs-Vermessung'. The top navigation bar includes 'Edit', 'Add comment', 'Assign', 'More', 'Refinement', and 'Waiting For Feedback'. Below the title, there are sections for 'Details' and 'Description'.

Details:

Type:	Epic	Status:	IN PROGRESS (View Workflow)
Priority:	Minor	Resolution:	Unresolved
Component/s:	None		
Labels:	Standort-FAIP Verpackungen Versand Versandverpackung		
Epic Name:	Aufbau Produktverpackungs-Vermessung		

Description:

Beschreibung:
Ich als IT und Versand moechte, dass die Abmessungen der Pakete an der Versandanlage automatisch erkannt werden. Die Abmessungen sollen außerdem einer Shipping-Nr. zugeordnet werden.

Akzeptanzkriterien:

Nr.	Akzeptanzkriterien	Status
1	Der Sensortraeger ist aufgebaut	✓
2	Die Daten werden erfasst	✓
3	Die Daten werden gespeichert	✓
4	Abmessungen und Shipping-Nr. werden zugeordnet	✓
5	RM an Marco Thiergart	✗

Attachments: Drop files to attach, or [browse](#).

Issues in epic:

Issue Key	Summary	Status	Assignee
FAIPSE-3459	PaMesAn - Aufbau des Sensortraegers	WAITING FOR FEEDBACK	Johannes Leyrer
FAIPSE-3453	PaMesAn - Datenverarbeitungworker	WAITING FOR FEEDBACK	Johannes Leyrer

Abbildung 5: Beispiel Epic

A.8 Architekturdesign

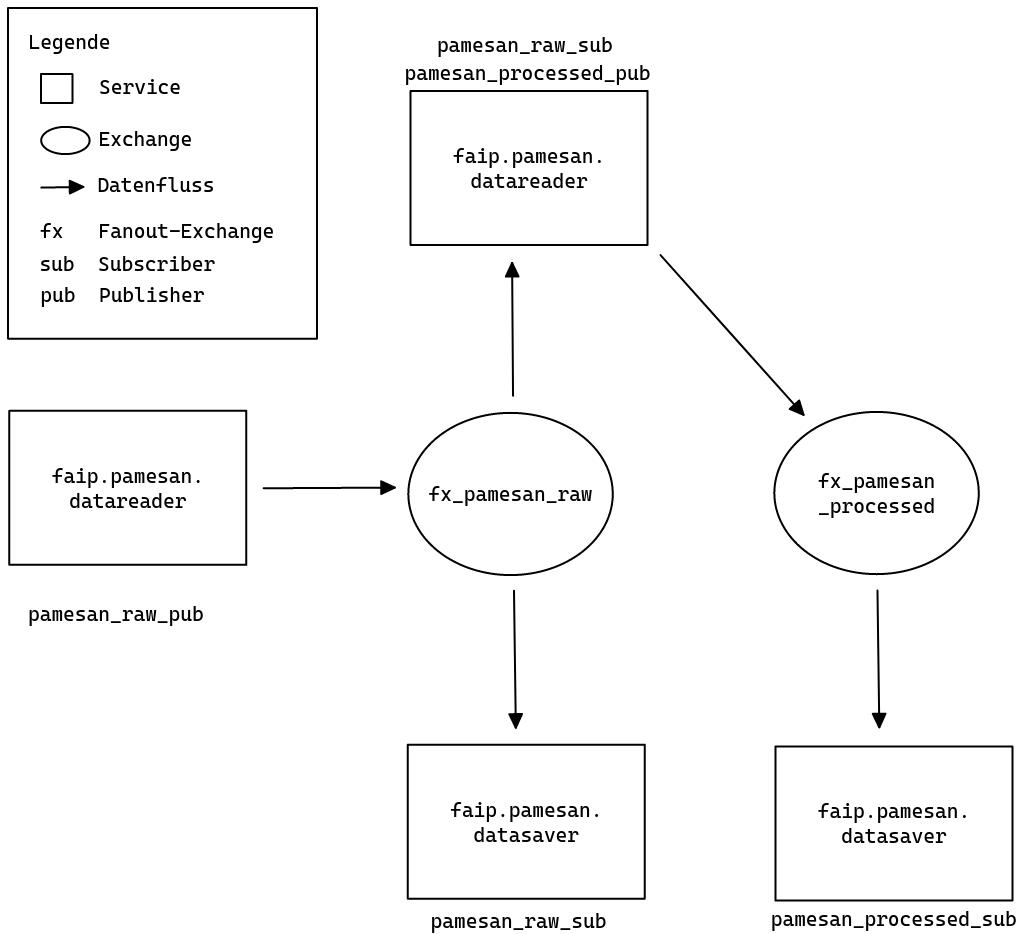


Abbildung 6: Skizze des Architekturdesigns

A.9 Tabellenmodell

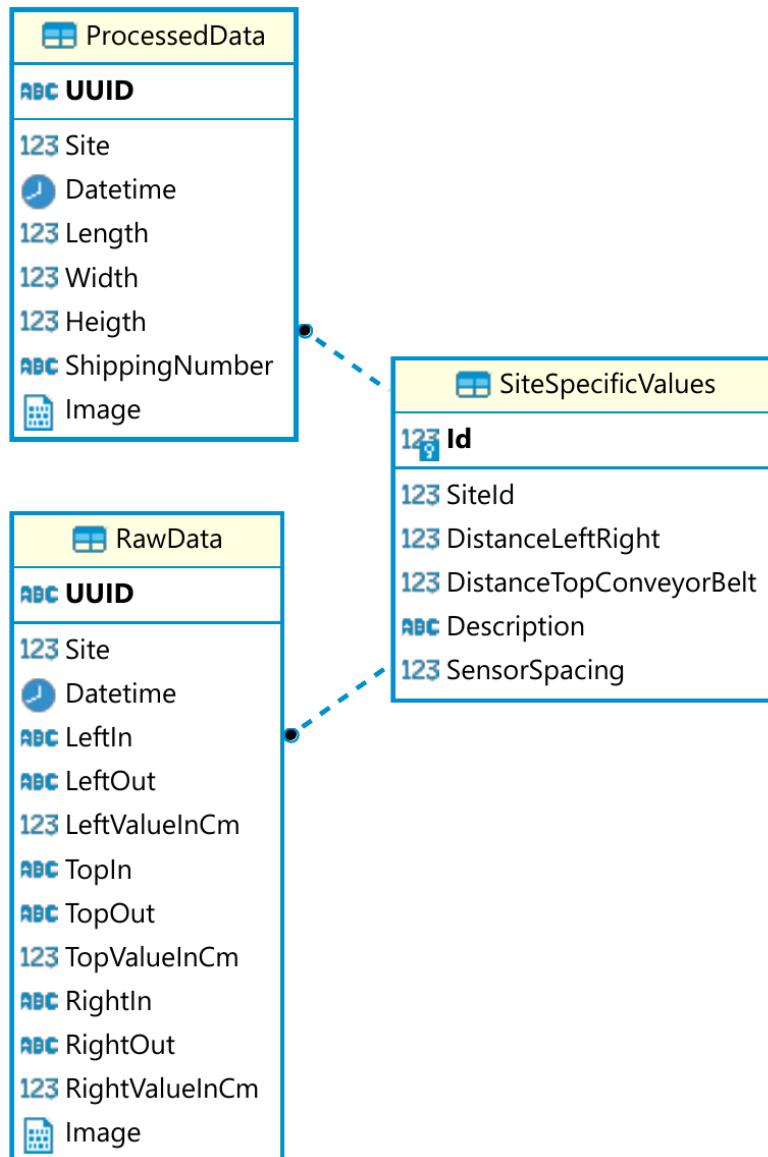


Abbildung 7: Tabellenmodell

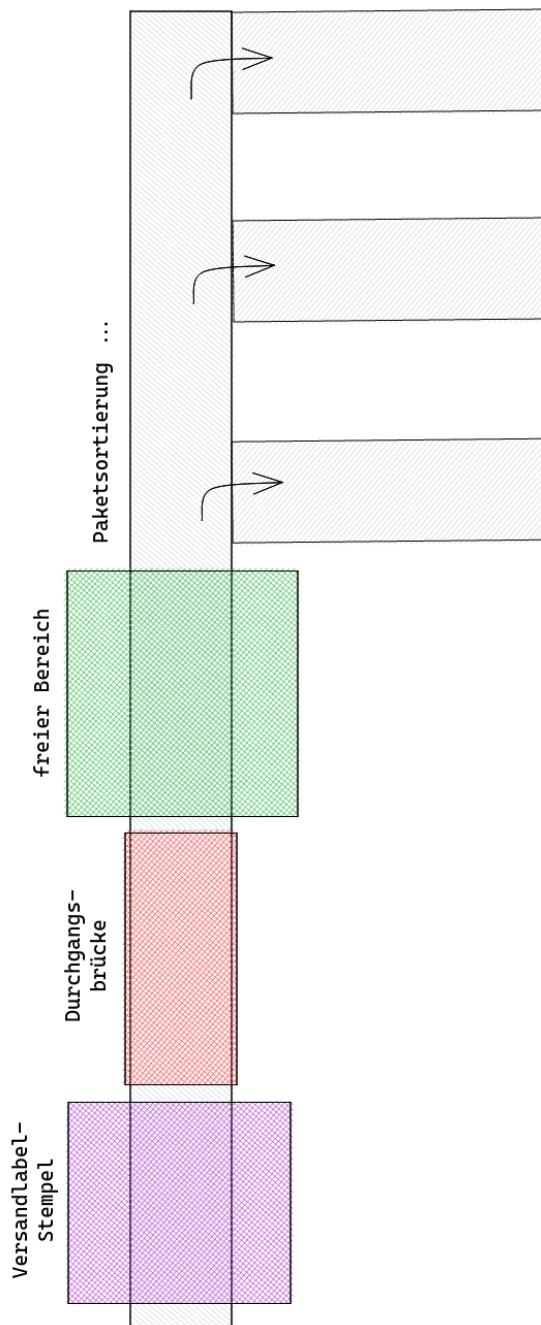
A.10 Skizze der Versandanlage

Abbildung 8: Skizze der Versandanlage

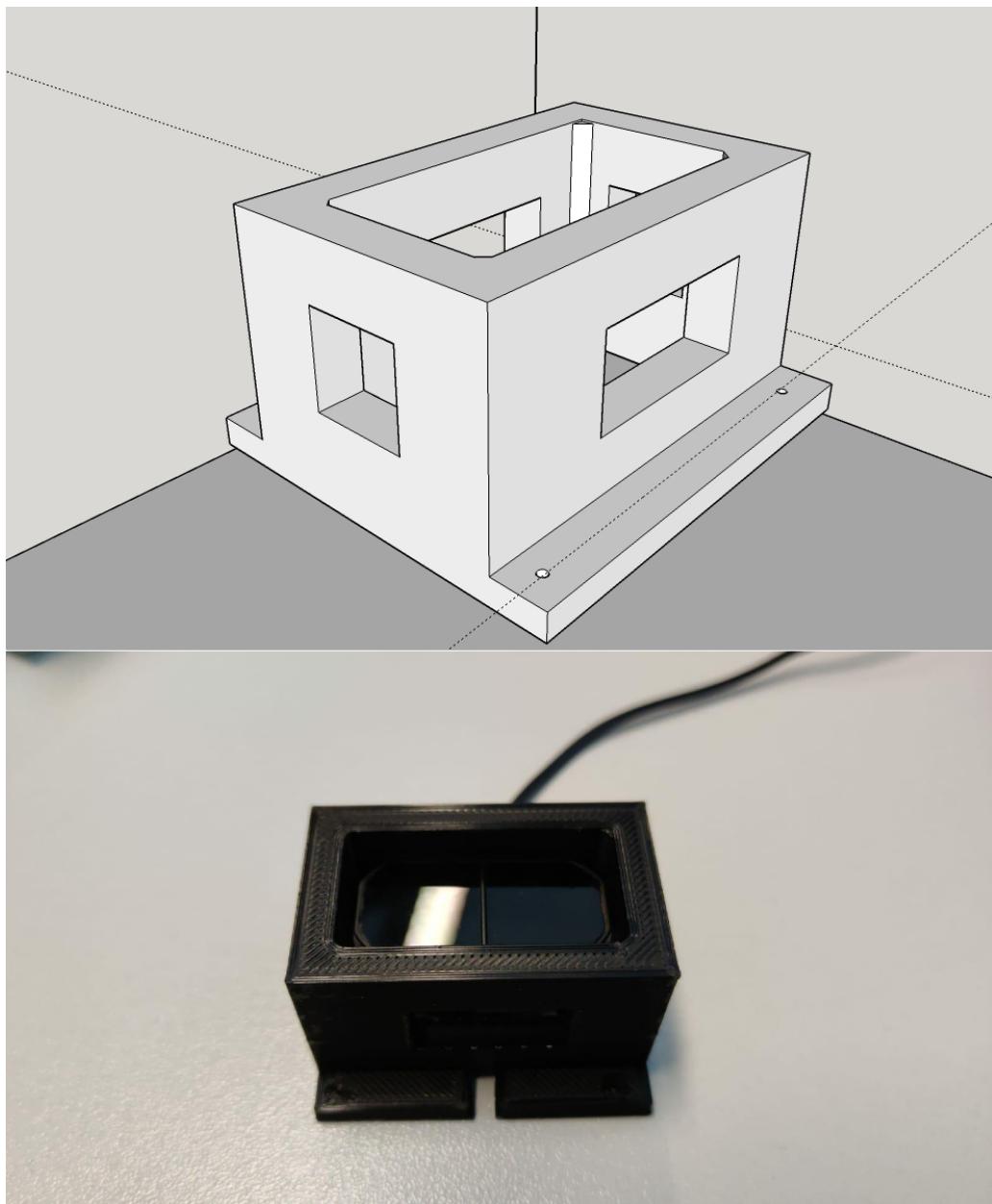
A.11 Sensorhalterung

Abbildung 9: Sensorhalterung: Vorlage [thingiverse] und zusammengesetztes Endprodukt

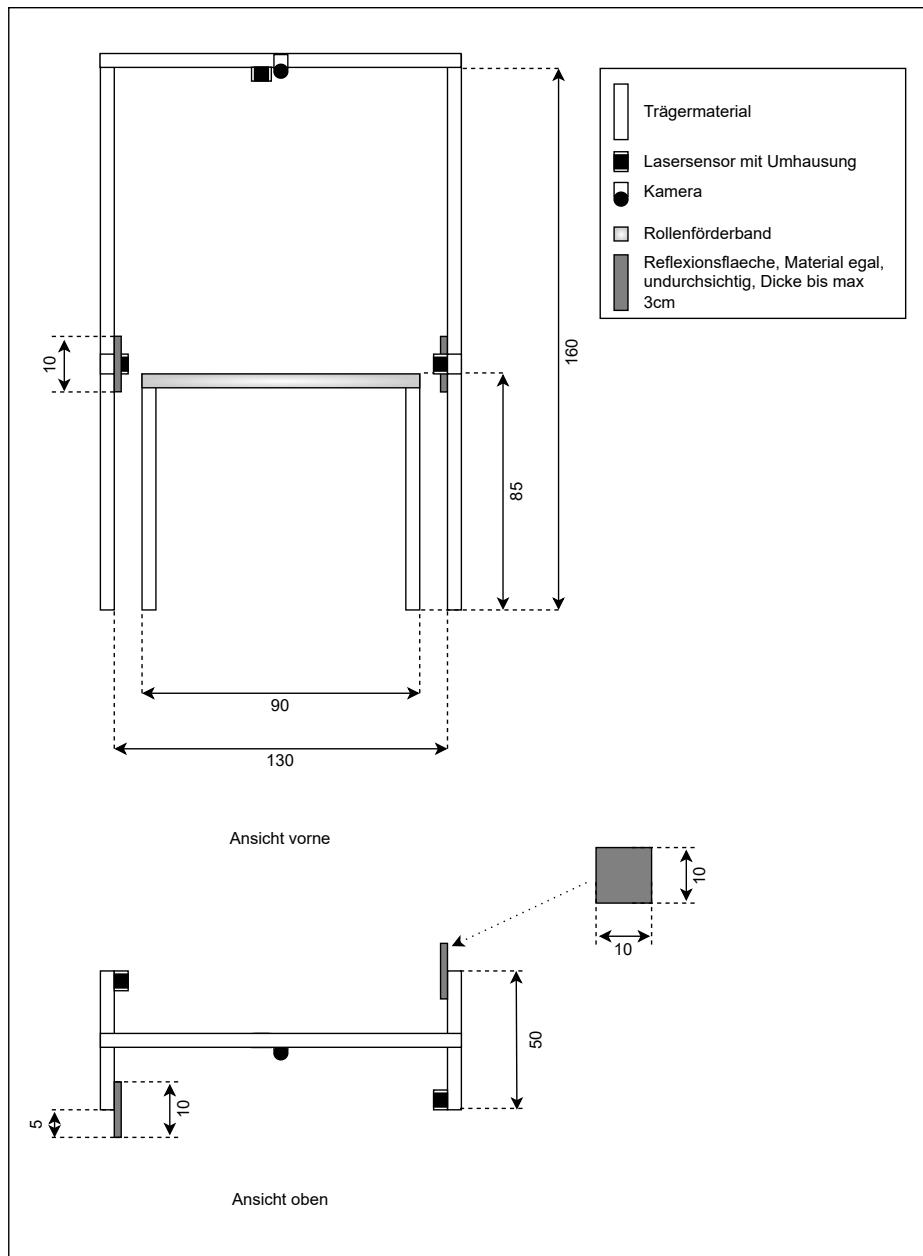
A.12 2D-Modell des Sensorträgers

Abbildung 10: 2D-Modell des Sensorträger

A.13 3D-Modell des Sensorträgers

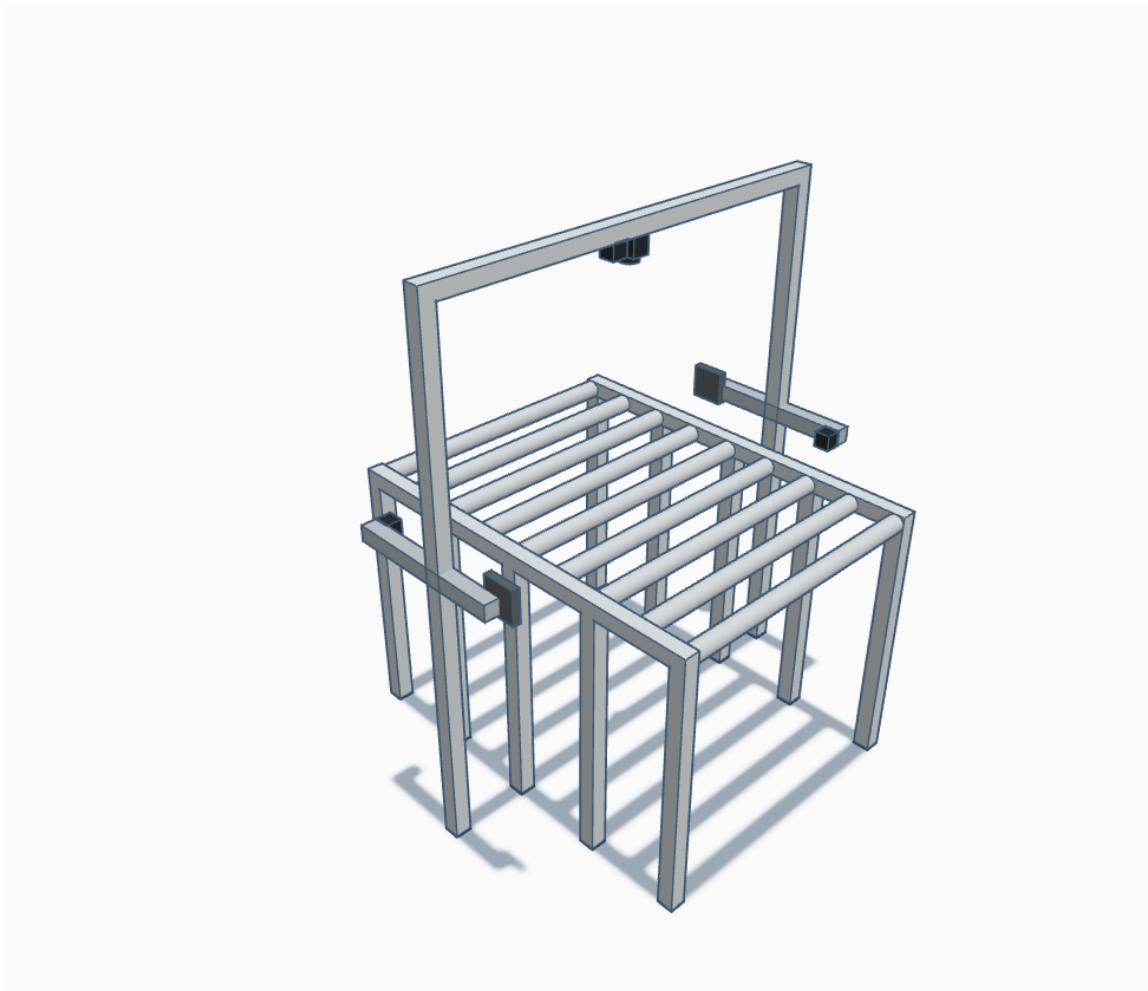


Abbildung 11: 3D-Modell des Sensorträgers

A.14 Fertiger Sensorträger



Abbildung 12: Fertig montierter und aufgestellter Sensorträger

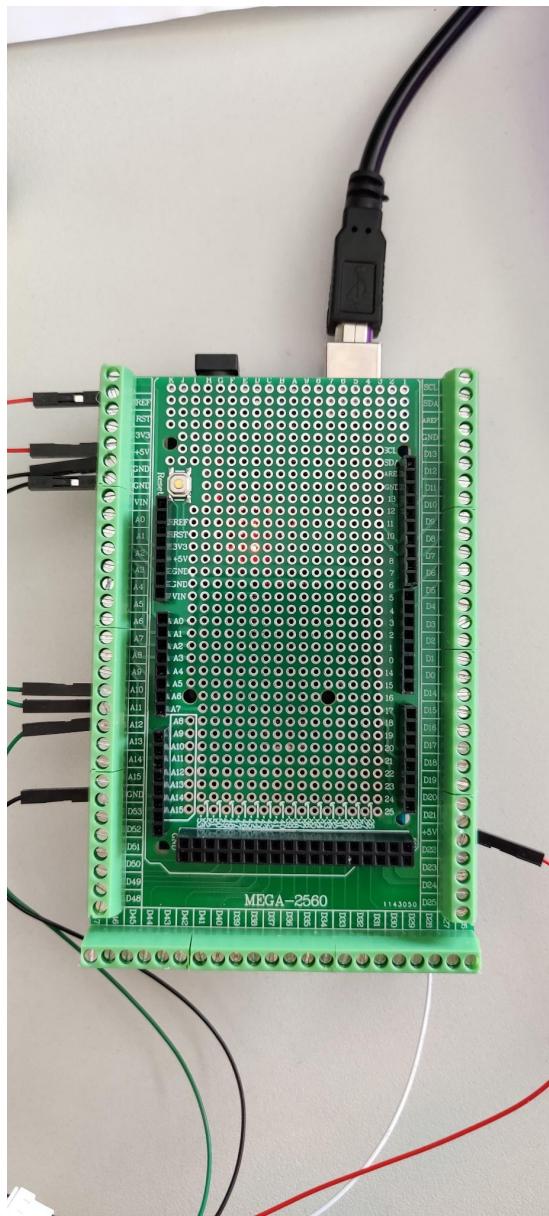
A.15 Arduino mit Lasersensoren

Abbildung 13: Arduino mit angeschlossenen Sensoren und Shield

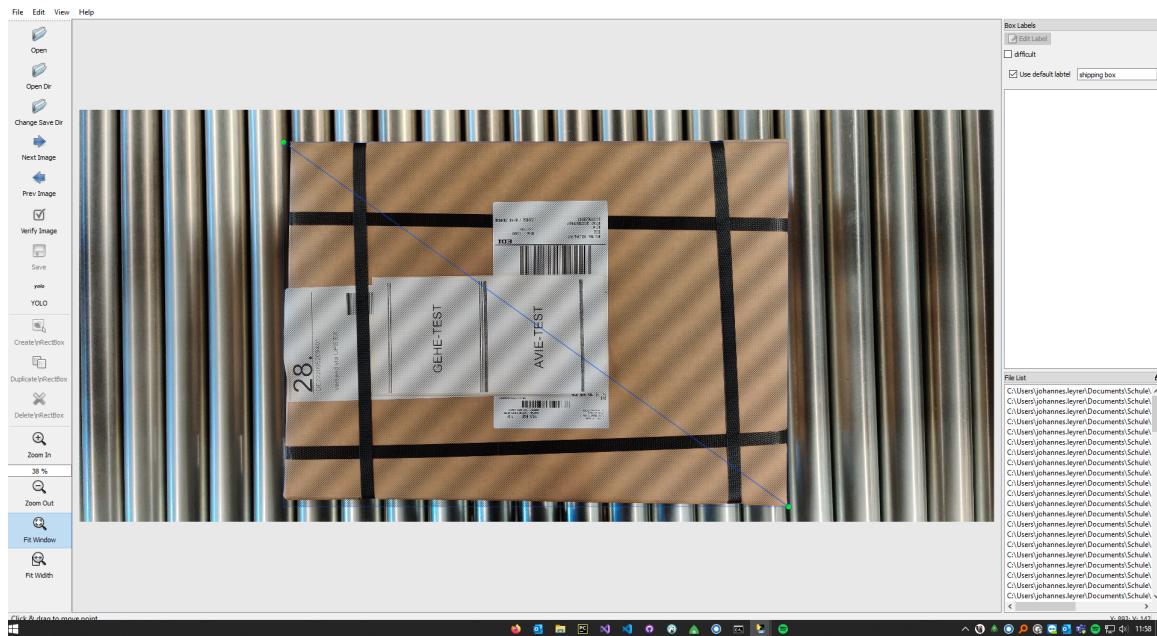
A.16 Beispiel Verwendung labelImg

Abbildung 14: Beispiel Verwendung labelImg

A.17 Beispiel Barcodeerfassung

Abbildung 15: Beispiel Barcodeerfassung

A.18 Beispiel Versandverpackungserkennung mit YOLOv7

Abbildung 16: Beispiel Versandverpackungserkennung mit Test-Label

A.19 Beispiel Labelerfassung



Abbildung 17: Beispiel Erfassung des Versandlabels mit Test-Label

A.20 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.21 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 Versandlabels 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 und das Versandlabel im Bild und berechnet die Abmessungen des Pakets.

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

B.1 Sensorträger und Komponenten

Der Sensorträger wurde anhand der in Abb. 18 auf Seite B-2 und Abb. 19 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-INTERN]\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-INTERN]\FAIP-Software\[NAME]\FAIP.PaMesAn\Sensortraeger\Sensoren`

Die Sensoren sind mit 4-adrig verschraubbaren Male-Sensorkabeln versehen, zu sehen in Abb. 20 auf Seite B-3. Die Anordnung ist bei Replikation unbedingt beizubehalten, um den Wartungsaufwand gering zu halten. Die ausgehenden Kabel sind wie in Abb. 21 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.

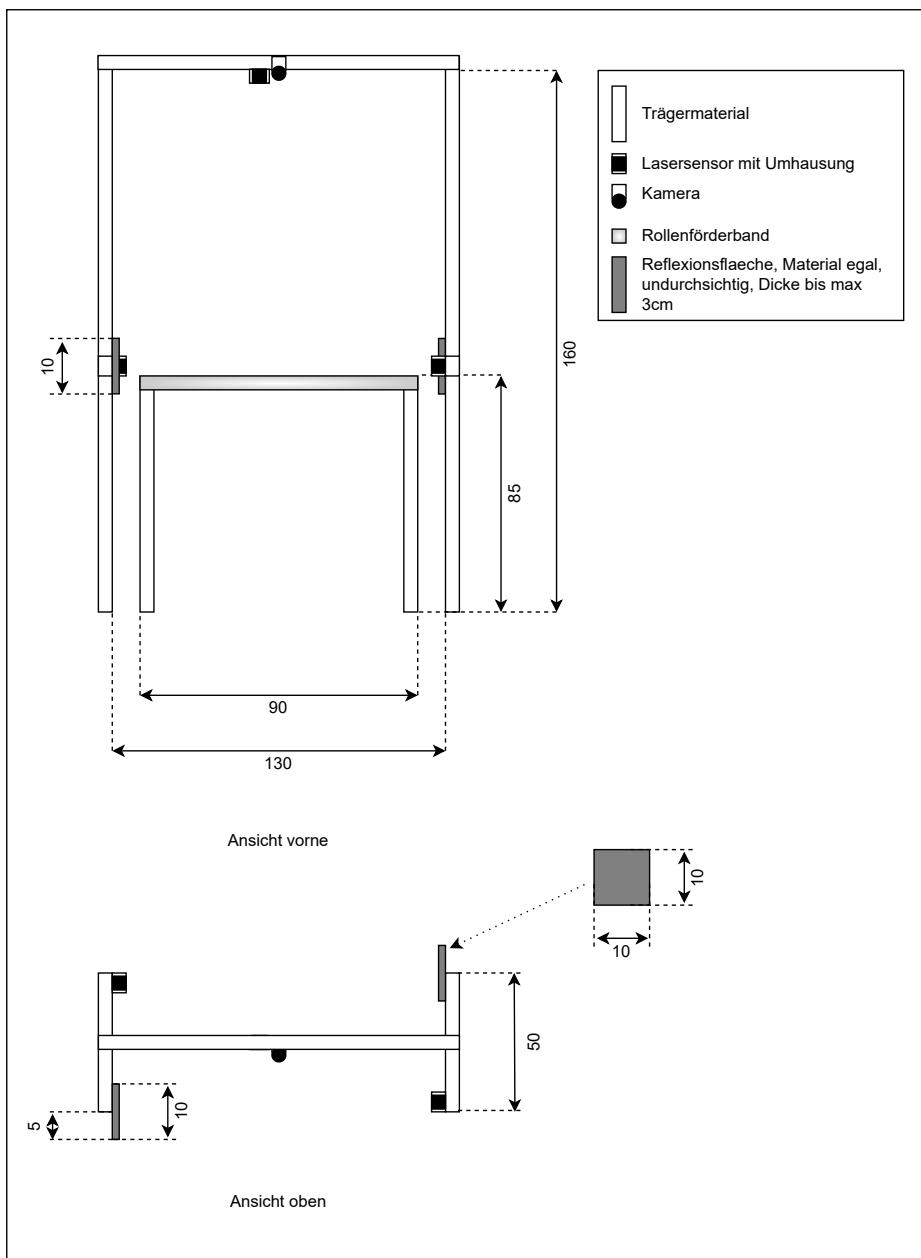


Abbildung 18: 2D-Zeichnung Sensorträger



Abbildung 19: 3D-Zeichnung Sensorträger

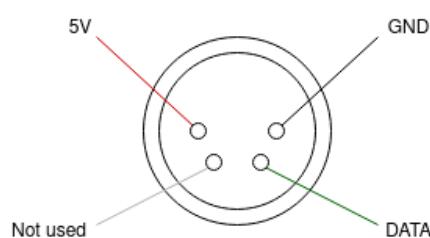


Abbildung 20: Anschlusspins der 4-adrig ver-schraubbaren Male-Sensorkabel

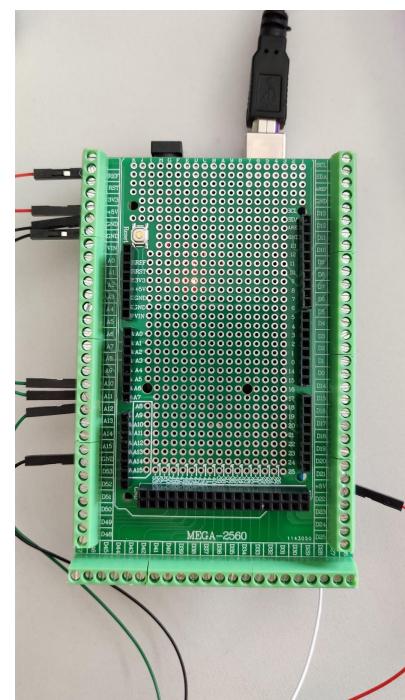


Abbildung 21: Pinbelegung am Arduino

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://\[GITLAB-INTERN\].de/\[NAME\]/faip.lib.rmq](https://[GITLAB-INTERN].de/[NAME]/faip.lib.rmq)

Der Service selbst ist unter folgender URL zu finden:

[https://\[GITLAB-INTERN\].de/\[NAME\]/faip.pamesan.datasaver](https://[GITLAB-INTERN].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 Rabbit-MQ übermittelt. 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://\[GITLAB-INTERN\].de/\[NAME\]/faip.pamesan.datareader](https://[GITLAB-INTERN].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 der 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 die 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 zur 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 übermittelt 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 Versandlabel und der darauf liegende Barcode erkannt und ausgelesen. Diese Daten werden dann über RabbitMQ veröffentlicht.

RMQ Diese Klasse stellt die Verbindung zum RabbitMQ-Server her, empfängt die rohen Sensor- und Bilddaten und versendet die berechneten Abmessungen, das ausgelesene Versandlabel 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 Versandlabel.

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
```