

Bakkalaureatsarbeit

# **Automatisierte Wissensgraph-Erstellung per Multi-Objekterkennung für einen KI-Wartungsassistenten**

bearbeitet von

Robin Morgenstern

geboren am 27.01.2004 in Chemnitz

Technische Universität Dresden

Fakultät Informatik  
Institut für Software- und Multimediatechnik  
Lehrstuhl Softwaretechnologie



Betreuer: Dr.-Ing. Karsten Wendt  
Hochschullehrer: Prof. Dr. rer. nat. habil. Uwe Aßmann

Eingereicht am 8. Juli 2025



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Motivation und Problemstellung . . . . .	1
1.2	Zielsetzung der Arbeit . . . . .	1
1.3	Forschungsfrage . . . . .	1
1.4	Aufbau der Arbeit . . . . .	1
<b>2</b>	<b>Theoretischer Hintergrund</b>	<b>3</b>
2.1	Wissensgraphen in der semantischen Datenverarbeitung . . . . .	3
2.2	Ontologien als formale Grundlage für Wissensgraphen . . . . .	3
2.3	Objekterkennung und Aufbau der annotierten Bilddateien . . . . .	3
2.4	Large Language Models und Verbindung zu Wissensgraphen . . . . .	4
2.5	Automatisierte Wissensgraphgenerierung - Stand der Technik . . . . .	4
<b>3</b>	<b>Methodik</b>	<b>7</b>
3.1	Struktur der Datenbasis . . . . .	7
3.2	Anforderung an die Graphstruktur . . . . .	8
3.2.1	Semantische und strukturelle Anforderungen . . . . .	8
3.2.2	Anforderungen an die technische und funktionale Nutzbarkeit . . . . .	8
3.3	Konzeption des automatischen Generierungsprozesses . . . . .	9
3.3.1	Ablaufmodell des Konvertierungsprozesses . . . . .	9
3.3.2	Logik zur Relationsermittlung . . . . .	9
3.4	Technische Umsetzung und Implementierungsdetails . . . . .	11
3.4.1	Strukturelle Varianten der Generierung . . . . .	11
3.4.2	Codebasierte Umsetzung der Varianten . . . . .	12
3.4.3	Erweiterung des Generierungssystems für mehrere Kameras . . . . .	13
3.4.4	Allgemeine Nutzbarkeit und Integration . . . . .	14
3.5	Validierungskonzept und Evaluationsmetriken . . . . .	14
<b>4</b>	<b>Evaluation</b>	<b>15</b>
4.1	Vergleich der erzeugten Wissensgraphstrukturen . . . . .	15
4.2	Analyse von Qualität, Vollständigkeit und Robustheit . . . . .	15
4.3	Integration und Verarbeitung durch verschiedene LLM-Models . . . . .	15
<b>5</b>	<b>Diskussion</b>	<b>17</b>
5.1	Interpretation der Evaluationsergebnisse . . . . .	17
5.2	Methodische Herausforderungen und Limitationen . . . . .	17
5.3	Einordnung in den aktuellen Forschungsstand . . . . .	17
5.4	Beantwortung der Forschungsfrage . . . . .	17
5.5	Perspektiven zur Weiterentwicklung und industriellen Anwendung . . . . .	17
<b>6</b>	<b>Literaturverzeichnis</b>	<b>i</b>

<b>A</b>	<b>Appendix</b>	<b>iii</b>
A.1	Additional Information . . . . .	iii
A.2	More Important Information . . . . .	iii

# **1 Einleitung**

## **1.1 Motivation und Problemstellung**

## **1.2 Zielsetzung der Arbeit**

## **1.3 Forschungsfrage**

## **1.4 Aufbau der Arbeit**



## 2 Theoretischer Hintergrund

### 2.1 Wissensgraphen in der semantischen Datenverarbeitung

Wissensgraphen (Knowledge Graphs, KGs) sind eine Form der strukturierten Wissensrepräsentation, die sich im Bereich der Künstlichen Intelligenz etabliert haben. Sie bilden durch Entitäten, deren Beziehungen sowie weiteren semantisch beschriebenen Merkmalen, komplexe Sachverhalte ab. Die Knoten eines KGs repräsentieren reale oder abstrakte Objekte. Die Kanten hingegen modellieren Relationen zwischen diesen Entitäten [1]. Wissensgraphen bieten wichtige Vorteile im Gegensatz zu anderen Arten der Wissensrepräsentation. So sind sie, durch ihr graphbasiertes Datenmodell, leicht verständlich und erlauben eine prägnante, intuitive Abstraktion. Sie besitzen zudem kein festes Schema, was eine flexible Entwicklung ermöglicht [2]. Wissensgraphen verbessern außerdem die Qualität von KI-Systemen, insbesondere für Frage-Antwort-Systeme, die eine besondere Rolle in dieser Arbeit spielen [3].

### 2.2 Ontologien als formale Grundlage für Wissensgraphen

Ontologien legen die Basis für eine formale Darstellung von Wissen. Sie fungieren dabei als eine Art Konvention oder Richtlinie [2]. Sie sind in der Lage, die Vereinheitlichung terminologischer Konzepte zu fördern und ein konsistentes Verständnis zwischen unterschiedlichen Domänen zu ermöglichen [4]. Eine Ontologie enthält Entitäten, Relationen, Eigenschaften sowie je nach Ontologiesprache auch Axiome [2, 4]. Mithilfe von taxonomischen Relationen können Konzept-Hierarchien zwischen Entitäten definiert werden. Die Beziehungen zwischen einzelnen Entitäten können mithilfe von sogenannte nicht-taxonomische Relationen (auch Object-Properties genannt) beschrieben werden [4]. Ontologien dienen als formales Schema für die Erstellung von Wissensgraphen, und können genauso wie Wissensgraphen als Graphstruktur modelliert werden [2, 3]. Somit sind Wissensgraphen als Instanz einer Ontologie mit Daten anzusehen [5].

### 2.3 Objekterkennung und Aufbau der annotierten Bilddateien

Objekterkennung (Object Detection) ist ein zentrales Aufgabenfeld im Bereich der Computer Vision. Das Hauptziel der Objekterkennung besteht darin, in einem Bild Instanzen von definierten Klassen zu identifizieren und deren genaue Position im Bild zu bestimmen [6]. Mit dem Aufkommen von Deeplearning im Jahr 2014 wurden viele verschiedene Architekturen von Convolutional Neural Networks (CNNs) entwickelt, die sich in zwei Kategorien einteilen lassen. Zwei-Stufen-Detektoren trennen die Objekterkennung und die Klassifizierung. Beispiele hiervon sind Varianten wie R-CNN, Fast R-CNN und Faster R-CNN. Ein-Stufen-Detektoren hingegen führen die Objektlokalisierung und -klassifizierung gleichzeitig in dem selben Durchlauf aus. In der Regel sind diese Detektoren schneller und damit besser geeignet für Echtzeitanwendungen (wie etwa in einem digitalen Wartungsassistenten). Beispiele hiervon sind YOLO (You Only Look Once) und SSD (Single-Shot Multibox Detector) Architekturen. Die Objekterkennung des

Digitalen Wartungsassistenten beruht auf der SSD-Architektur, da diese versucht die Geschwindigkeit von YOLO mit der Genauigkeit von Zwei-Stufen-Methoden zu vereinen, indem sie Techniken der beiden Ansätze kombiniert [6]. Diese Positionen können auf unterschiedliche Weise bestimmt werden, aber die Option, die im Laufe dieser Arbeit eine Rolle spielt, sind Bounding-Boxes. Dabei ermittelt die Objekterkennung den Begrenzungsrahmen, die Objektklasse sowie einen Konfidenzwert (der die geschätzte Wahrscheinlichkeit für die Klassenzuordnung angibt) [6]. Diese Informationen werden in einer CSV-Datei gespeichert. Bei den annotierten Bilddateien, die im Laufe der Arbeit verwendet werden, handelt es sich um den Output eines trainierten Objekterkennungsmodells. Diese CSV-Datei soll nun in einen Wissensgraph überführt werden, damit LLMs mithilfe dieser Daten Fragen zu den Maschinen beantworten können.

### 2.4 Large Language Models und Verbindung zu Wissensgraphen

Large Language Models (LLMs) sind Künstliche Intelligenz (KI) Sprachmodelle, die viele Milliarden Parameter umfassen und typischerweise auf der Transformer-Architektur basieren [7, 8]. Dabei fungieren sie als generative mathematische Modelle, die auf Textdaten im Umfang von mehreren hundert Terabyte trainiert wurden [7]. Diese Modelle berechnen die statistischen Wahrscheinlichkeiten für die möglichen Fortsetzungen einer Wortsequenz und sagen so neue Token voraus – diese können einzelne Zeichen, Wortteile oder ganze Wörter sein [7]. Somit basieren die Antworten von LLMs nur auf statistischen Korrelationen und weisen kein menschliches Verständnis auf. Aus diesem Grund verfügen LLMs auch über kein tatsächliches Wissen oder Verständnis von Wahrheit oder Falschaussagen [7]. Dies kann dazu führen, dass Halluzinationen (irrelevante oder unsinnige Aussagen, die allerdings natürlich sind) auftreten [7]. Wissensgraphen können dem LLM "Wissen injizieren", um somit wissensbasierte Anwendungen zu ermöglichen, welche logische Schlussfolgerungen erlauben. Aus diesen Gründen werden Wissensgraphen bereits in Frage-Antwort-Systemen eingesetzt [3, 1, 9]. Da der digitale Wartungsassistent ebenfalls ein Frage-Antwort-System darstellt, sind LLMs und ihr Umgang mit Wissensgraphen von großer Bedeutung für diese Arbeit. Die Wissensgraphen werden aus visuellen Daten generiert und ermöglichen die Beantwortung einfacher (einzeln Tripel) sowie komplexerer Abfragen. Diese können über mehrstufiges Schlussfolgern (multi-hop) anhand mehrerer Tripel des Wissensgraphen beantwortet werden [3]. Außerdem tragen die implementierten Wissensgraphen zur Lösung des Black-Box-Problems bei, da man mithilfe dieser die Herkunft einer Antwort erklären und nachvollziehen kann, was das Vertrauen in die Vorhersagen stärken kann [1]. Die Kombination vereint somit die Fähigkeit von Wissensgraphen zur Speicherung und Abfrage von Wissen mit der Ausdrucksstärke von LLMs [1, 9, 7, 8].

### 2.5 Automatisierte Wissensgraphgenerierung - Stand der Technik

Die automatisierte Wissensgraphgenerierung – auch als Ontology Learning bezeichnet – ist der Prozess der Erstellung oder Erweiterung von Wissensgraphen durch automatische oder semiautomatische Verfahren [4]. Kernmodule der Generierung liegen in der Wissensextraktion sowie der Wissensverlinkung [9]. Die manuelle Erstellung von Wissensgraphen ist ein arbeitsintensiver und zugleich komplexer Prozess [10]. Bei großen Domänen kann es sogar die menschlichen Fähigkeiten übersteigen [4]. Diesen manuellen Prozess versucht die automatische Wissensgraphgenerierung zu minimieren bzw. zu umgehen. Der Stand der Technik umfasst dabei eine Vielzahl an Ansätzen und Technologien. Natural Language Processing (NLP) wird zur Analyse und Extraktion von Informationen aus Texten verwendet [4]. Dies umfasst Techniken wie Named



Entity Recognition (NER), Relation Extraction oder die Analyse lexico-syntaktischer Muster [4, 9, 10]. Auch Methoden des maschinellen Lernens (ML) werden umfassend eingesetzt [9]. Aktuelle Forschung untersucht zudem den Einsatz von LLMs zur Extraktion, Schemaerstellung und Abfragegenerierung im Bereich des Knowledge Graph Engineering (KGE) [5]. Die Konstruktion multimodaler Wissensgraphen ist allerdings weiterhin ein herausforderndes Problem [3]. Bislang sind nur wenige Arbeiten bekannt, die sich mit einem Wissensgraphgenerator aus visuellen Quellen befassen. Genau an diesem Punkt setzt diese Arbeit an, und untersucht, ob ein solcher Generator umsetzbar und effizient ist.



## 3 Methodik

### 3.1 Struktur der Datenbasis

#### Format und Aufbau der Annotationsdateien

Die Wissensgraph-Erstellung soll aus der Output-Datei eines Multi-Objekterkennungsmodell erfolgen. Dabei erhält jedes Bild eine eigene .csv Datei mit allen erkannten Objekten, die im Rahmen dieser Arbeit die erkannten Bauteile (Components) der Industriemaschinen darstellen. CSV-Dateien (Comma-Separated Values) sind Klartextdateien, welche jeden Datensatz, dessen Felder durch Kommas getrennt sind, in einer Zeile repräsentieren. Die Nutzung dieser Dateien bietet sich im Zusammenhang dieser Arbeit an, da diese eine einfache Struktur und eine breite Unterstützung bieten [11]. Die Spalten der Dateien enthalten die Detection Scores (Erkennungswerte, welche angeben wie sicher sich das Objekterkennungsmodell ist), Classes (Klassen der erkannten Bauteile), sowie die Werte x min, y min, x max und y max. Anhand dieser 4 Koordinaten lassen sich die Bounding Boxes der Bauteile herleiten, die es später ermöglichen, räumliche Relationen zwischen den erkannten Objekten abzuleiten. Diese Koordinaten sind pixelbasiert und beziehen sich direkt auf das Bildformat (in unserem Fall 512x512 Pixel). Die Objektklassen sind konsistent in englischer Sprache benannt (z. B. Motor, Switch, Lamp).

Detection Score	Class	x min	y min	x max	y max
0.927	Lamp	222	74	316	132
0.859	Grinding wheel	39	141	147	253
0.932	Grinding wheel	319	130	409	246
0.969	Motor	117	170	349	218
0.849	Switch	220	242	262	264
0.727	Fuse	143	299	196	350

Tabelle 3.1: Beispielhafter Ausschnitt einer CSV-Datei

#### Aufbau der Datenbasis

Die Datenbasis besteht aus insgesamt neun annotierten Bilddateien, die jeweils verschiedenen Perspektiven auf drei Industriemaschinen zeigen. Bei den Maschinen handelt es sich um eine Schleifmaschine, eine Bohrmaschine und um einen Schalter, die zwischen vier und sieben Bauteile aufweisen. Der Aufbau der CSV-Dateien orientiert sich hierbei am typischen Output eines Objekterkennungsmodells der SSD-Architektur. Da diese Modelle nicht immer fehlerfrei arbeiten, wurde der Ablauf innerhalb der Datenbasis simuliert und die Daten manuell erstellt. Dadurch kann die Richtigkeit und Vollständigkeit der Annotationen sichergestellt werden, was eine Grundlage für die nachfolgenden Schritte der Wissensgraphgenerierung bietet.



Abbildung 3.1: Visualisierte Boundingboxes des Beispiels:

## 3.2 Anforderung an die Graphstruktur

### 3.2.1 Semantische und strukturelle Anforderungen

Die Zielstruktur der generierten Wissensgraphen ist ein gerichteter RDF-Graph nach dem OWL-Standard. Dieser Repräsentiert Wissen durch Tripel, welche aus Subjekt - Prädikat - Objekt bestehen (z. B. Lamp above Motor) [3, 9, 12]. Jedes erkannte Bauteil soll als Individuum der OWL-Klasse Components instanziiert werden. Da alle Objekte eines Wissensgraphen eindeutig benannt werden müssen, müssen gleiche Bauteile anhand ihrer Position im Bild, von links nach rechts, nummeriert werden. Jeder Wissensgraph erhält außerdem ein Individuum Machine der OWL-Klasse Machines, durch welches man später die relative Position der Bauteile zur gesamten Maschine definieren kann. Die Relationen des Wissensgraphen werden auf OWL-Properties (Objekt-Properties und Data-Properties) abgebildet. Anhand der Relationen left\_to, right\_to, above und below sollen sich die Richtungsabhängigkeit zwischen den einzelnen Individuen beschreiben lassen. Die Relationen inside\_of und outside\_of sollen topologische (raumumschließende) Relationen zwischen den einzelnen Individuen modellieren. Außerdem soll mithilfe der in\_the\_middle\_of Relation die relative Mitte der Maschine als Orientierungspunkt auf das mittig platzierte Bauteil der Maschine abgebildet werden. Dabei sollen inverse Relationen implementiert werden (also left\_to - right\_to, above - below, inside\_of - outside\_of), da diese für die Vollständigkeit des Wissensgraphen essentiell sind. Die RDF-Struktur soll dabei nicht auf eine Mindest- oder Maximalanzahl an Objekten begrenzt sein und theoretisch später anschlussfähig an domänspezifische Wissensgraphen mit faktenbasiertem Hintergrundwissen sein, die Informationen zu den Eigenschaften und Funktionen der jeweiligen Bauteile enthält.

### 3.2.2 Anforderungen an die technische und funktionale Nutzbarkeit

Der generierte Wissensgraph soll die etablierten Standards des semantischen Webs entsprechen, vor allem den Spezifikationen von RDF und OWL. Um eine maschinelle Weiterverarbeitung zu ermöglichen, soll der Wissensgraph sowohl als OWL-Datei, als auch in einem Tripeltextformat gespeichert werden können. Für die Einbindung in LLM-basierte Frage-Antwort-Systeme ist

zudem erforderlich, dass die Relationen aussagekräftige und semantisch eindeutige Prädikate verwenden. Dies ermöglicht insbesondere mehrstufiges Schlussfolgern (multi-hop reasoning) auf Basis von logischen Ketten zwischen Entitäten. Darüber hinaus muss die Struktur des Graphen regelkonform und widerspruchsfrei sein. Das betrifft vor allem:

- die Verwendung von eindeutigen Namenskonventionen für Klassen, Individuen und Properties,
- die Vermeidung widersprüchlicher Relationen (z. B. A above B und B above A),
- die Einhaltung syntaktischer Regeln der OWL/RDF-Spezifikation,
- sowie die Unterstützung von inversen Relationen (da diese bei Richtungsangaben von großer Bedeutung sind)

Diese Anforderungen stellen sicher, dass der erzeugte Wissensgraph sowohl interoperabel und flexibel, als auch funktional für die Integration in wissensbasierte KI-Systeme geeignet ist.

## 3.3 Konzeption des automatischen Generierungsprozesses

### 3.3.1 Ablaufmodell des Konvertierungsprozesses

Ziel dieses Prozesses ist die automatische Übersetzung strukturierte Annotationsdaten (CSV) in einen OWL-konformen RDF-Graphstruktur. Dafür beginnt der Prozess im ersten Schritt mit dem Einlesen der Annotationsdaten. Dabei werden auch weitere Werte berechnet, wie `x_center` und `y_center`, welche die Mittelpunkte der Bounding Boxes darstellen. Anhand dieser Daten werden nun die einzelnen Bauteile als OWL-Individuen der Klasse `Components` instanziiert. Um die OWL und RDF Standards einzuhalten und Mehrfachbenennung zu umgehen, werden gleiche Bauteile nummeriert. Diese Nummerierung geschieht von links nach rechts im Bild anhand der `x_center` Koordinaten. Zusätzlich wird ein Individuum `Machine` der Klasse `Machines` erzeugt. Anschließend werden die Data Properties `x_minimum`, `y_minimum`, `x_maximum`, `y_maximum`, `x_center`, `y_center` sowie `detection_score` den jeweiligen Instanzen zugewiesen. Im nächsten Schritt werden nun mithilfe regelbasierter Algorithmen die räumlichen Relationen abgeleitet (siehe 3.3.2. Anhand dieser abgeleiteten Relationen werden nun die RDF-Tripel erstellt (z. B. `Lamp above Motor`). Am Ende wird die erzeugte Graphstruktur in einer OWL-Datei gespeichert, sowie durch einen extra Script in eine Textdatei aus Klartext-Tripeln konvertiert. Für eine visualisierte Version, siehe 3.2.

### 3.3.2 Logik zur Relationsermittlung

Um die Ableitung semantisch interpretierbarer Relationen aus rein geometrischen Daten zu ermöglichen, wurden regelbasierte Algorithmen definiert. Diese verwenden die Koordinaten der Bounding Boxes, sowie die daraus berechneten Mittelpunktkoordinaten (`x_center` und `y_center`). Die Bestimmung der horizontalen Relationen wie `left_to` und `right_to` basiert auf den Differenzen der `x_center` und `y_center` Werte zweier Individuen. Dabei dient das Vorzeichen der x-Differenz zur Richtungszuordnung. Um zu vermeiden, dass Bauteile horizontale Relationen erhalten, obwohl sie stark versetzt entlang der y-Achse sind, wird zusätzlich das Verhältnis der y- zur x-Differenz betrachtet. Die Relation wird nur gesetzt, wenn das Verhältnis einen bestimmten Schwellenwert nicht übersteigt. Dafür prüft der Algorithmus folgende Schrägheitsbedingung:

$$|y_i - y_j| < \frac{1}{\text{threshold}} \cdot |x_i - x_j|$$

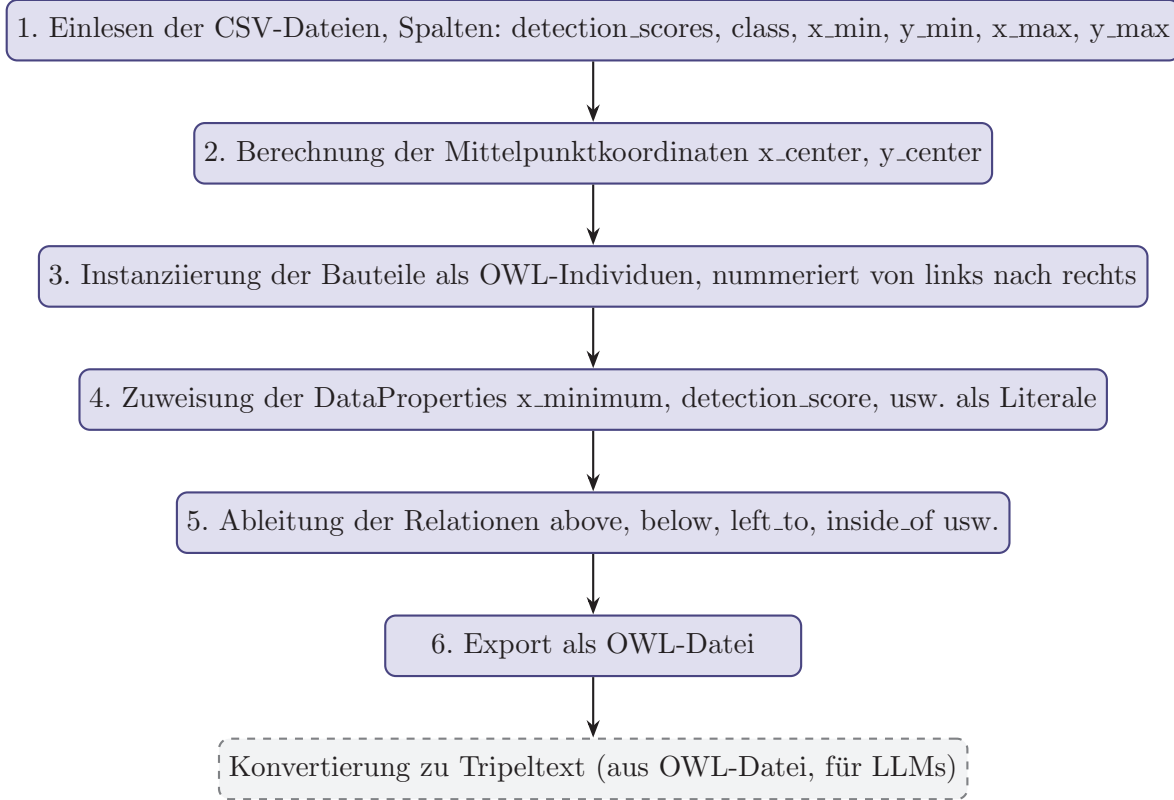


Abbildung 3.2: Flussdiagramm des Ablaufs

Die besten Ergebnisse wurden mit einem Wert von  $\text{threshold} = 0,7$  empirisch ermittelt, was bedeutet, dass das Nachbar-Bauteil maximal eine Abweichung von ca.  $55^\circ$  von der horizontalen Achse des betrachtenden Bauteils abweichen darf (0 Grad heißt hierbei perfekt horizontal). Dies verhindert, dass Bauteile als links oder rechts erkannt werden, obwohl sie fast vertikal unter dem Bauteil angeordnet sind. Der akzeptierte Toleranzbereich für horizontale Relationen liegt somit zwischen 0 und 55 Grad, sowohl nach oben als auch nach unten. Hierbei ist es also möglich, dass ein Individuum, z. B. sowohl links, als auch unter einem anderen Bauteil liegen kann. Durch die Verwendung von gerichteten Differenzen ist allerdings sichergestellt, dass keine widersprüchlichen Relationen (z. B. A left\_to B und B left\_to A) entstehen können, da das Vorzeichen der Differenz eindeutig bestimmt, in welcher Richtung die Bauteile voneinander liegen. Optional kann der Algorithmus sowohl zwischen impliziter und expliziter Erkennung unterscheiden: Im impliziten Modus wird nur die Relation des nächstgelegenen Bauteils in jede Richtung berücksichtigt, während im expliziten Modus alle zulässigen Richtungsrelationen gesetzt werden. Die vertikalen Relationen above und below werden analog zu den horizontalen Relationen bestimmt, wobei die x- und y-Differenzen entsprechend vertauscht werden.

Die topologischen Relationen wie inside\_of und outside\_of werden anhand der minimalen und maximalen Koordinaten der Eckpunkte der Bounding Boxes hergeleitet ( $x_{\min}$ ,  $y_{\min}$ ,  $x_{\max}$ ,  $y_{\max}$ ). Dabei berechnet der Algorithmus die Überschneidung zweier Bounding Boxes und setzt diese ins Verhältnis zur Fläche der größeren Box:

$$\frac{\text{Area}(\text{Box}_i \cap \text{Box}_j)}{\max(\text{Area}(\text{Box}_i), \text{Area}(\text{Box}_j))} > \text{threshold}$$

Überschreitet dieser Wert einen definierten Schwellenwert (ebenfalls empirisch mit `threshold = 0,7` bestimmt), wird das kleinere Objekt als `inside_of` dem größeren Individuum als Relation zugeordnet.

Für die Relation `in_the_middle_of` wird das Bauteil bestimmt, dessen Mittelpunktkoordinaten dem Durchschnitt der Mittelpunktkoordinaten aller Bauteile am nächsten liegt. Das eingesetzte Framework `owlready2` ermöglicht außerdem die Definition inverser Operationen, wodurch jede Relation automatisch eine semantisch äquivalente Gegenrelation erzeugt. Die beschriebenen Algorithmen gewährleisten nicht nur Konsistenz und Widerspruchsfreiheit, sondern erzeugen auch verständliche und reproduzierbare Ergebnisse.

## 3.4 Technische Umsetzung und Implementierungsdetails

### 3.4.1 Strukturelle Varianten der Generierung

Um die Umsetzung des Generierungsprozesses zu ermöglichen wurde ein modulares Programmsystem entwickelt. In diesem Kapitel werden Da der Forschungsstand zu Wissensgraphen mit visuellen Informationen sehr begrenzt ist, ist die Implementierung und Analyse unterschiedlicher Architekturvarianten von besonderer Bedeutung. Denn je nach Anwendungsfall und Datenlage könnten unterschiedliche Varianten optimal sein.

**Richtungsrelationen - implizit oder explizit:** Der Wissensgraphgenerator unterstützt zwei Varianten zur Erzeugung von Richtungsrelationen: eine implizite und eine explizite Struktur. Bei der impliziten Relationserzeugung wird immer nur die Richtungsrelation des nächsten Nachbarbauteils gesetzt. Dies gilt je Richtung, sodass jedes Individuum maximal vier Richtungsrelation besitzen kann (von jeder Relation eine). Dadurch wird der Graph kompakter und deutlich weniger komplex, was insbesondere im Einsatz mit kleineren LLMs in einem Frage-Antwort-Model dafür sorgen könnte, dass es den Wissensgraph besser verarbeiten kann. Allerdings müsste diese `multi-hop-reasoning` einsetzen, um z. B. alle Bauteile unter der Lampe zu nennen. Bei der expliziten Relationserzeugung hingegen werden alle zulässigen Richtungsrelation gesetzt, sodass jedes Individuum theoretisch maximal  $n-1$  Relationen erhalten könnte ( $n$  = Anzahl der Individuen). Hierbei würde der Graph deutlich komplexer, aber auch semantisch vollständiger. Auch das `multi-hop-reasoning` wird dadurch vermieden, allerdings könnte ein LLM es schwieriger haben, herauszufinden welches Bauteil beispielsweise direkt unter der Lampe ist.

**Numerische Daten - mit oder ohne Koordinaten:** Auch im Bezug auf numerische Informationen besitzt der Wissensgraphgenerator zwei Möglichkeiten: ohne Koordinaten und mit Koordinaten. Ohne die Speicherung der Koordinaten wird der erzeugte Wissensgraph deutlich kompakter, was wie bereits erwähnt bei kleineren LLMs mit begrenzten Kontextfenster von großer Bedeutung sein könnte. Im Modus mit Koordinatenspeicherung werden diese in den Data-Properties `x_minimum`, `y_minimum`, `x_maximum`, `y_maximum`, `x_center` und `y_center` hinterlegt. Dadurch wird der Graph nachvollziehbarer und LLMs können beispielsweise räumliche Abstände abschätzen, ihre eigene Schlussfolgerungen ableiten oder Ergebnisse überprüfen.

**Exportformate - OWL und Tripeltext:** Auch der Exportprozess unterstützt zwei verschiedene Formate: OWL-Dateien und Textdateien im Tripel-Format. Wobei die OWL-Datei für den standardkonformen Export semantischer Graphen dient und Kompatibilität mit Ontologie-Editoren

wie z. B. Protege ermöglicht. Allerdings kann es für LLMs unvorteilhafter sein mit diesen Dateien umgehen, als mit einfachem Textinput. Für diese mögliche Verbesserung der Ergebnisse für die Integration in LLM Frage-Antwort-Systeme ist außerdem der Export in eine Textdatei der RDF-Tripel des erzeugten Graphen.

All diese Varianten erlauben eine flexible Generierung der RDF-Struktur je nach gewünschtem Zielsystem. Alle Varianten können dabei beliebig per Konfigurationsdatei kombiniert werden, sodass sich der jeweils geeignetste Architekturstil des Wissensgraphen je nach Zielsystem bestimmen lässt (siehe 4 zur LLM-Integration).

#### 3.4.2 Codebasierte Umsetzung der Varianten

Das Python Skript `OntologyGenerator.py` wurde unter der Verwendung des `owlready2` Frameworks entwickelt und stellt den zentralen Baustein der automatischen Generierung RDF-Wissensgraphen dar. Zunächst wird die Konfigurationsdatei eingelesen, deren Dateipfad beim Starten des Programms mit dem Argument `-c` übergeben wurde (z. B. `-c config.ini`). Aus dieser Datei werden die Parameter `output_path`, `csv_path`, `add_coordinates`, `explicit_mode`, `remove_false` und `summarize_graph` geladen.

Je nachdem, ob der `csv_path` eine CSV-Datei oder ein Ordner ist, wird der Single- oder der Multi-Cam-Modus aktiviert. Beim Single-Cam-Modus wird die CSV-Datei ausgelesen und das Skript berechnet bereits, welches Bauteil sich in der Mitte der Maschine befindet. Im Anschluss wird die Ontologie, die festen Klassen, die Object- und Data Properties sowie die inversen Properties mithilfe von `owlready2` erstellt. Danach wird für jeden Eintrag der CSV-Datei ein neues Individuum der Klasse `Components` erstellt und ihm der zugehörige Detection Score zugeordnet. Falls der Parameter `add_coordinates` in der Configdatei auf `true` gesetzt ist, werden zusätzlich ihre Koordinaten ergänzt. Anschließend werden den Individuen ihre räumlichen Relationen zugeordnet. Dies geschieht, indem für jedes Individuum, der in 3.3.2 beschriebene Algorithmus durchgeführt wird. Dieser unterscheidet allerdings anhand der `explicit_mode` Flag, ob die Relationen implizit und explizit modelliert werden. Beim impliziten Modus der `check_vertical` Funktion werden die Werte `above_min` und `below_min` mit dem Wert 1024 initialisiert (da das der maximale Abstand ist, der bei einem 512x512 Bild möglich ist) und die leeren Listen `above_class` und `below_class` erstellt. Dabei wird für jedes Bauteil, das links oder rechts von dem Individuum liegt, der Abstand berechnet und verglichen, ob dieser unter dem Minimalabstand liegt. Wenn nein, ignoriert der Algorithmus das Bauteil, da das nähere Bauteil bereits in der Liste stehen muss. Wenn ja, werden die Einträge der Liste gelöscht, und das neue Bauteil wird der Liste hinzugefügt. Dabei wird der neue Minimalabstand in die `left_min` oder `right_min` Variable geschrieben. So kann der Algorithmus zuverlässig das nächstgelegene Bauteil für jede Richtung bestimmen. Beim expliziten Modus läuft das Programm analog ab, außer dass die Listenelemente nicht entfernt werden und der Mindestabstand auf 1024 festgelegt bleibt. Dadurch können in die Liste mehrere Bauteile aufgenommen werden und ermöglichen so die explizite Modellierung. Der Algorithmus für die vertikalen Relationen läuft genauso ab und muss deswegen nicht nochmal erklärt werden.

Anschließend wird die Relation `in_the_middle_of` zwischen dem bereits am Anfang bestimmten mittleren Bauteil und dem Objekt `Machine` gesetzt. Danach wird die Funktion `check_inside` aufgerufen, welche die topologischen Relationen modelliert. Die Funktionsweise dieser wurde allerdings bereits in 3.3.2 ausführlich erleutert und weist keine wesentlichen codebasierten Besonderheiten auf. Das `owlready2` Framework erkennt erst beim Speichern der OWL-Datei Doppelungen und inverse Relationen, was zu Problemen führen kann, da der Code mit den aktuellen Werten der `owlready2`-internen Relationslisten weiterarbeitet (vor allem im Multi-Cam-Modus,



siehe 3.4.3). Aus diesem Grund werden anschließend die zwei Hilfsfunktionen `reverse_properties` und `remove_redundant_properties` aufgerufen. Dabei ergänzt die `reverse_properties` Funktion für jedes Tripel aus Subjekt - Prädikat - Objekt, das inverse Tripel aus Objekt - inverses Prädikat - Subjekt (z. B. Lamp above Motor - Motor below Lamp). Da bei dieser Funktion auch Doppelungen auftreten können, werden diese im Anschluss mit der `remove_redundant_properties` Funktion entfernt, indem die owlready2-Relationenlisten in ein Set und danach wieder in eine Liste umgewandelt werden. Da ein Set per Definition keine Duplikate enthalten kann, werden dadurch alle doppelten Einträge gelöscht.

Am Ende des Programms wird die RDF-Graphstruktur als OWL-Datei exportiert und kann, wenn notwendig, mit dem `StatementGenerator.py` Skript in eine Textdatei mit Tripeln umgewandelt werden. Dafür muss das Programm mit dem `-o` Parameter gestartet werden, welcher den Pfad der OWL-Datei angibt. Durch die Konfigurationsdatei ist es möglich, die verschiedenen Varianten miteinander beliebig zu kombinieren und diese jederzeit einfach zu ändern, was kombinierte Tests und Architekturvergleiche ermöglicht. Durch die modulare Struktur des Programms lässt es sich leicht erweitern und einzelne Komponenten können unabhängig voneinander getestet werden.

#### 3.4.3 Erweiterung des Generierungssystems für mehrere Kameras

Die Wissensgraph-Erstellung aus einem Kamerawinkel ist für diesen optimiert, aber kann wichtige Informationen nicht enthalten. Denn je nach Blickwinkel können bestimmte Komponenten versteckt, gar nicht sichtbar in toten Winkeln oder von der Objekterkennung nicht erkannt worden sein. Um diese Probleme zu beheben, wurde der Generierungsprozess um einen Modus für mehrere Kameras (Multi-Cam-Modus) ergänzt. Durch ihn ist der Prozess in der Lage, verschiedene Perspektiven zu modellieren und so dem LLM Frage-Antwort-System zusätzlichen Kontext zur Verfügung stellen.

Der Multi-Cam-Modus wird gestartet indem der `csv_path` Parameter der Konfigurationsdatei auf einen Ordner mit mehreren CSV-Dateien zeigt und läuft analog zum Single-Cam-Modus ab, aber besitzt einige große Unterschiede. Dafür werden zuerst alle CSV-Dateien aus dem Ordner gefiltert. Danach führt der Prozess die bereits in 3.4.2 beschriebenen Schritte durch, allerdings unterscheiden sich in ein paar Implementierungsentscheidungen. Dabei wird zu Beginn des Einlesens jeder CSV-Datei, eine neue Klasse `Camera_[Name der CSV-Datei]` erstellt. Die `Components` und `Machines` Klassen werden dann innerhalb dieser `Camera` Klasse erstellt. Diese Klassen werden in einem Python Dictionary gespeichert und können so jederzeit aufgerufen werden. Nachdem das Erstellen der Klassen durchgeführt wurde, läuft der Prozess wie im Single-Cam-Modus (siehe 3.4.2) ab und wiederholt sich für jede CSV-Datei. Dabei wird das Dictionary aller `Individuals` einer Iteration, innerhalb eines Dictionary `all_individuals` gespeichert, welches am Ende des letzten Durchlaufs alle `Individuals` enthält.

Da nun nach erfolgreichem Durchlaufen des Prozesses ein verschachteltes Dictionary anstatt einem Einfachen vorliegt, müssen die `check_inside`, `reverse_properties` und `remove_redundant_properties` Funktionen dafür angepasst werden. Dafür wurden die `check_inside_all`, `reverse_properties_all` und `remove_redundant_properties_all` Funktionen implementieren, welche sich um die Organisation dieses verschachtelten Dictionarys kümmern. Dabei lesen die Funktionen nacheinander jedes Einfache Dictionary aus dem Verschachtelten, führen die Ursprungsfunktion aus und ersetzen das jeweilige Einzeldictionary mit dem Aktualisieren. Dabei führen alle Änderungen an der Ursprungsfunktion auch zu einer Änderung der zugehörigen `_all` Funktionen, da diese wie eine Art Vermittler fungieren.

Nachdem sowohl die `check_inside_all` Funktion zur Bestimmung der topologischen Relationen

und die `reverse_properties_all` Funktion zum aktualisieren der owlready2-internen Relationslisten ausgeführt wurden, wird nun die `same_individuals` Funktion ausgeführt. Das Ziel dieser ist es, anhand eines regelbasierten Algorithmus zu erkennen, welche Bauteile der unterschiedlichen Kameras die identischen Bauteile sein könnten. Dabei soll er die OWL-native `sameAs` Relation zu den identischen Individuen hinzufügen. Dabei rechnet diese Funktion den Grad der Ähnlichkeit anhand der gleichen Relationen aus. Dafür werden die Individuals mit allen anderen Individuals der anderen Kameras verglichen und vergleicht sie paarweise. Zur Berechnung wird eine Ähnlichkeitsformel benutzt, deren Bestandteile im Folgenden definiert werden:

- $A, B$ : Zu vergleichende Individuen
- $R$ : Menge aller zu betrachteten Relationen (z. B. *above*, *below*, *left\_to*, ...)
- $r(A)$ : Menge aller Ziel-Individuen, mit denen  $A$  über die Relation  $r$  verknüpft ist
- $|M|$ : Anzahl der Elemente in der Menge  $M$

$$\text{similarity}(A, B) = \frac{\sum_{r \in R} |r(A) \cap r(B)|}{\sum_{r \in R} |r(A)|}$$

$$\text{threshold} = \begin{cases} \text{threshold}_{\text{equal}}, & \text{wenn } \text{name}(A) = \text{name}(B) \\ \text{threshold}_{\text{diff}}, & \text{wenn } \text{name}(A) \neq \text{name}(B) \end{cases}$$

$$\text{mit} \quad \begin{cases} \text{threshold}_{\text{equal}} = \text{threshold} - \text{tolerance} \\ \text{threshold}_{\text{diff}} = \text{threshold} \end{cases}$$

$$\text{Dann gilt: } \text{similarity}(A, B) > \text{threshold} \quad \Rightarrow \quad A \equiv B$$

#### 3.4.4 Allgemeine Nutzbarkeit und Integration

### 3.5 Validierungskonzept und Evaluationsmetriken

## **4 Evaluation**

**4.1 Vergleich der erzeugten Wissensgraphstrukturen**

**4.2 Analyse von Qualität, Vollständigkeit und Robustheit**

**4.3 Integration und Verarbeitung durch verschiedene LLM-Models**



## **5 Diskussion**

**5.1 Interpretation der Evaluationsergebnisse**

**5.2 Methodische Herausforderungen und Limitationen**

**5.3 Einordnung in den aktuellen Forschungsstand**

**5.4 Beantwortung der Forschungsfrage**

**5.5 Perspektiven zur Weiterentwicklung und industriellen  
Anwendung**

## 6 Literaturverzeichnis

- [1] Shaoxiong Ji u. a. “A Survey on Knowledge Graphs: Representation, Acquisition, and Applications”. In: *IEEE Transactions on Neural Networks and Learning Systems* 33 (2 Feb. 2022), S. 494–514. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2021.3070843.
- [2] Aidan Hogan u. a. “Knowledge Graphs”. In: *ACM Computing Surveys* 54.4 (Juli 2021), S. 1–37. ISSN: 1557-7341. DOI: 10.1145/3447772.
- [3] Ciyuan Peng u. a. “Knowledge Graphs: Opportunities and Challenges”. In: *Artificial Intelligence Review* 56.11 (Apr. 2023), S. 13071–13102. ISSN: 1573-7462. DOI: 10.1007/s10462-023-10465-9.
- [4] Lan Yang, Kathryn Cormican und Ming Yu. “Ontology Learning for Systems Engineering Body of Knowledge”. In: *IEEE Transactions on Industrial Informatics* 17 (2 Feb. 2021), S. 1039–1047. ISSN: 1941-0050. DOI: 10.1109/TII.2020.2990953.
- [5] Christian Zinke-Wehlmann und Julia Friedrich, Hrsg. *First Working Conference on Artificial Intelligence Development for a Resilient and Sustainable Tomorrow. AI Tomorrow 2023*. Informatik Aktuell Series. 4 Soft and Missing Spots of Human-Centered AI Implementation. Wiesbaden, Germany: Springer Vieweg, 2024. 1158 S. ISBN: 9783658437053.
- [6] Ravpreet Kaur und Sarbjeet Singh. “A comprehensive review of object detection with deep learning”. In: *Digital Signal Processing* 132 (Jan. 2023), S. 103812. ISSN: 1051-2004. DOI: 10.1016/j.dsp.2022.103812. URL: <http://dx.doi.org/10.1016/j.dsp.2022.103812>.
- [7] Murray Shanahan. “Talking about Large Language Models”. In: *Communications of the ACM* 67.2 (Jan. 2024), S. 68–79. ISSN: 1557-7317. DOI: 10.1145/3624724. URL: <http://dx.doi.org/10.1145/3624724>.
- [8] Haiyan Zhao u. a. “Explainability for Large Language Models: A Survey”. In: *ACM Transactions on Intelligent Systems and Technology* 15.2 (Feb. 2024), S. 1–38. ISSN: 2157-6912. DOI: 10.1145/3639372. URL: <http://dx.doi.org/10.1145/3639372>.
- [9] Sanju Tiwari, Fatima N. Al-Aswadi und Devottam Gaurav. “Recent trends in knowledge graphs: theory and practice”. In: *Soft Computing* 25.13 (Apr. 2021), S. 8337–8355. ISSN: 1433-7479. DOI: 10.1007/s00500-021-05756-8. URL: <http://dx.doi.org/10.1007/s00500-021-05756-8>.
- [10] Christopher Brewster u. a. “Issues in learning an ontology from text”. In: *BMC Bioinformatics* 10.S5 (Mai 2009). ISSN: 1471-2105. DOI: 10.1186/1471-2105-10-s5-s1.
- [11] Stephan Mäs u. a. “Generic schema descriptions for comma-separated values files of environmental data”. In: *The 21th AGILE International Conference on Geographic Information Science*. 2018.
- [12] Heiner Ludwig, Thorsten Schmidt und Mathias Kühn. “An ontology-based retrieval augmented generation procedure for a voice-controlled maintenance assistant”. In: *Computers in Industry* 169 (Aug. 2025), S. 104289. ISSN: 0166-3615. DOI: 10.1016/j.compind.2025.104289. URL: <http://dx.doi.org/10.1016/j.compind.2025.104289>.



# A Appendix

## A.1 Additional Information

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

## A.2 More Important Information

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.



## **Erklärung**

Ich erkläre, dass ich die vorliegende Arbeit selbstständig, unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Dresden, den 8. Juli 2025