Hochschule Pforzheim

Fakultät für Technik

Wirtschaftsingenieurwesen

Stromverbrauchsanalyse mit OCR

Bachelor-Thesis

1. Gutachter: Prof. Dr. Raphael Volz
2. Gutachter: Prof. Dr. Heiko Thimm
vorgelegt von: Lukas Vaessen
vorgelegt am: 11.09.2023

Inhaltsverzeichnis

Eidesstattliche Erklärung

Ich versichere, die beiliegende Thesis selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benützt sowie alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet zu haben. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Pforzheim,11.09.23
Ort/Datum Unterschrift

Kurzfassung

Diese Arbeit präsentiert die Entwicklung eines kosteneffizienten Systems zur automatischen Erfassung und Analyse von Zählerständen. Das System nutzt Mikrocontroller mit integrierter Kamera und Optical Character Recognition (OCR) Technologie. Drei verschiedene Mikrocontroller wurden getestet: ESP32-EYE, M5Stack UnitV2 und TimerCamX. Die TimerCamX wurde aufgrund ihrer technischen Eigenschaften, Flexibilität und Kosteneffizienz als beste Lösung ausgewählt.

Die OCR-Engine 2 von OCR.space wurde für die Texterkennung verwendet. Sie ist besonders gut auf Zahlen in Bildern trainiert und stellt einen kostenlosen Dienst dar, der Text aus Bildern und gescannten Dokumenten extrahiert. Um die Genauigkeit der erfassten Zählerstände zu verbessern, wurden Prüflogiken und Plausibilitätsprüfungen im Code implementiert.

Trotz einiger Herausforderungen, wie der Anpassung der Beleuchtungsverhältnisse, der Positionierung der Kamera und dem Schreiben des Codes, konnte das System erfolgreich implementiert und getestet werden. Die Ergebnisse zeigen, dass die automatische Erfassung von Zählerständen durch den Einsatz von kostengünstigen Mikrocontrollern und OCR-Technologie möglich und effektiv ist.

Zukünftige Arbeiten könnten sich auf die Verbesserung der Lichtverhältnisse und die Integration von maschinellem Lernen konzentrieren, um die Genauigkeit der Texterkennung weiter zu verbessern. Maschinelles Lernen könnte dabei helfen, die OCR-Ergebnisse zu optimieren und die Fehlerrate zu reduzieren.

1. Einleitung

In dieser Arbeit wird die Anwendbarkeit der digitalen Zeichenerkennung (OCR) zur Analyse des Stromverbrauchs untersucht. Angesichts der wachsenden Bedeutung von Nachhaltigkeit und Energieeffizienz stellt die genaue Messung und Analyse des Stromverbrauchs eine zentrale Herausforderung dar. Diese Herausforderung wird durch die zunehmende Digitalisierung und Vernetzung unserer Gesellschaft noch verstärkt.

Im Mittelpunkt steht die Anwendung der OCR-Technologie zur Umwandlung von abfotografierten Texten und Zahlen in maschinenlesbaren Text, speziell in Bezug auf Daten aus analogen Stromzählern. Die zentrale Forschungsfrage lautet: Wie kann die OCR-Technologie zur Analyse von Daten aus analogen Stromzählern angewendet werden und welche OCR-Engines und Hardwareoptionen sind dafür am besten geeignet?

Die Untersuchung erfolgt im Kontext der Hochschule Pforzheim und unter Berücksichtigung von Budgetbeschränkungen, um aufzuzeigen, wie solche Lösungen mit geringen Mitteln umgesetzt werden können. Die in dieser Arbeit erarbeiteten Methoden und Technologien könnten in Zukunft auch auf private Haushalte angewendet werden.

Die Arbeit baut auf einem Vorprojekt im Fach Cyber Physical Systems auf, in dem eine komplexere Lösung mit Home Assistant verwendet wurde. Im Rahmen dieser Arbeit wird ein praktischer Versuchsaufbau erstellt und implementiert. Die daraus resultierenden Daten werden analysiert und diskutiert, um ein umfassendes Verständnis der Stärken und Schwächen der vorgeschlagenen Ansätze zu gewinnen.

Im weiteren Verlauf der Arbeit wird ein finaler Aufbau am Zählerschrank der Hochschulbibliothek realisiert. Dieser dient als praktische Anwendung und Demonstration der erarbeiteten Methoden und Technologien.

2. Motivation

Die Energiepreise haben in den letzten Jahren ein historisches Hoch erreicht und stellen für viele Haushalte eine erhebliche finanzielle Belastung dar. Wie aus Abbildung 1 hervorgeht, stiegen die Strompreise in Deutschland in den letzten vier Jahren um beeindruckende 57,65 %, von 29,36 €/kWh auf 46,25 €/kWh. Abbildung 1

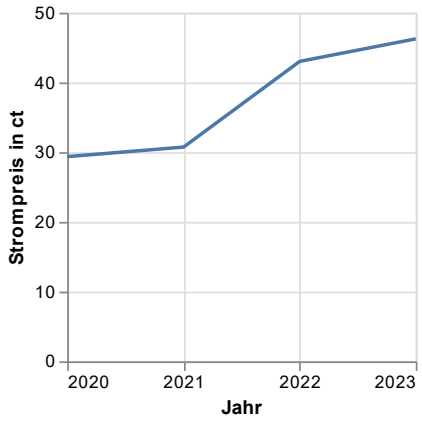


Abbildung 1 Strompreis 2020-2023[1][2]

Diese beträchtliche Preiserhöhung wurde durch globale Ereignisse wie die Corona-Pandemie und den Krieg in der Ukraine weiter verstärkt. Angesichts dieser Entwicklungen gewinnt die Energieeinsparung an Relevanz, nicht zuletzt auch unter dem Aspekt der Nachhaltigkeit.

Zur Reduzierung des Energieverbrauchs kann der Austausch konventioneller Leuchtmittel gegen energieeffiziente Alternativen wie LEDs beitragen, ebenso wie der Einsatz technischer Geräte zur Messung des Stromverbrauchs. Allerdings fokussieren viele dieser Geräte sich lediglich auf einzelne Verbrauchsquellen im Haushalt und ermöglichen daher keine umfassende Einsicht in den Gesamtverbrauch.

Smart Meter, bestehend aus einer digitalen Messeinrichtung und einem Gateway, könnten eine Lösung für eine umfassende Überwachung des Verbrauchs bieten. Jedoch ist diese Technologie oft mit hohen Installations- und Betriebskosten verbunden, was sie für kleinere Einrichtungen häufig unrentabel macht.

3. Grundlagen Verbrauchsmessung

In diesem Kapitel werden die Methoden zur Messung des Stromverbrauchs aufgezeigt. Für Strom gibt es im Wesentlichen drei gebräuchliche Messstandards: den analogen Zähler, den digitalen Zähler und das Smart Meter.

3.1 Analoge Zähler

Analoge Stromzähler, auch Ferrariszähler genannt, sind elektromechanische Instrumente, die auf der Grundlage der elektromagnetischen Induktion arbeiten, um den Stromverbrauch zu ermitteln und zu dokumentieren. Ein solcher Ferraris-Zähler ist mit einer rotierenden Scheibe ausgestattet, die durch den durchfließenden elektrischen Strom angetrieben wird. Die Drehbewegung der Scheibe wird von einem Zählwerk registriert, welches die Anzahl der Umdrehungen festhält und dadurch den Stromverbrauch erfasst. [3, 4]

Abbildung 2 analoger Stromzähler [5]

3.2 Digitaler Zähler

Ein digitaler Stromzähler, auch als elektronischer Stromzähler bezeichnet, verwendet elektronische Bauteile und Software zur Messung des Stromverbrauchs. Im Gegensatz zum elektromechanischen Ferrariszähler misst ein digitaler Stromzähler den Stromfluss nicht durch eine rotierende Scheibe, sondern durch die Erfassung von elektrischen Signalen. Diese Signale werden von Sensoren erfasst und an einen Mikrocontroller weitergeleitet, der den Stromverbrauch über einen bestimmten Zeitraum berechnet, in der Regel in Kilowattstunden (kWh). Die erfassten Daten werden in einem internen Speicher des Zählers gespeichert und können über verschiedene Schnittstellen abgerufen werden. [6, 7]

[7]

Abbildung 3 digitaler Stromzähler [5]

3.3 Smart-Meter

Ein Smart-Meter, auch als intelligenter Stromzähler bezeichnet, ist ein digitaler Stromzähler mit bidirektionaler Kommunikationsschnittstelle. Im Gegensatz zu herkömmlichen Stromzählern, die lediglich den Stromverbrauch messen, ermöglicht ein Smart Meter die Übertragung von Verbrauchsdaten an den Energieversorger und gibt dem Nutzer eine detaillierte Kontrolle über seinen Stromverbrauch. Die Messung des Stromverbrauchs mit einem Smart Meter erfolgt ähnlich wie bei einem digitalen Zähler durch Sensoren und Mikrocontroller. Allerdings werden die erfassten Daten nicht nur im Gerät gespeichert, sondern auch an den Energieversorger und den Nutzer weitergeleitet. Darüber hinaus bietet ein Smart-Meter die Möglichkeit, den Stromverbrauch in Echtzeit zu überwachen und zu optimieren. Diese Daten stehen dem Verbraucher jedoch nicht kostenfrei zur Verfügung, sondern müssen über Drittanbieter bezogen und ausgewertet werden. [6, 7]

Abbildung 4 Smart Meter [8]

4. Technische Grundlagen der Stromverbrauchsanalyse mit OCR

Dieses Kapitel beleuchtet die technischen Grundlagen der Stromverbrauchsanalyse mittels optischer Zeichenerkennung. Es werden verschiedene OCR-Engines wie Tesseract, Amazon Web Services und dieOCR.space API vorgestellt. Zudem werden relevante Hardwarekomponenten wie das ESP32-EYE, M5Stack UnitV2 TimerCamX untersucht. Die Geräte und OCR-Engines werden hinsichtlich ihrer Anwendbarkeit für die Stromverbrauchsanalyse bewertet.

4.1 Die optische Zeichenerkennung

Die optische Zeichenerkennung (OCR) stellt eine transformative Technologie dar, die Text aus Bildern extrahiert und in maschinenlesbare Text konvertiert. Dieses Kapitel konzentriert sich auf die verschiedenen Phasen der OCR, die weit über die bloße Erkennung von Zeichen hinausgehen. Es beginnt mit der Vorverarbeitung, die die Qualität des Eingangsbildes verbessert, gefolgt von der Segmentierung und Charakterisierung, die für die Identifizierung und Unterscheidung von Zeichen unerlässlich sind. Anschließend wird die Erkennungsphase betrachtet in welcher das Zeichen endgültig identifiziert wird. Dabei können Neuronale Netzte zum Einsatzkommen oder klassische Maschine Learning Ansätze wie Entscheidungsbäume, Support Vector Machine oder k-Nächste-Nachbarn-Schätzer(kNN). [9, 10]

4.2 Grundeinstellungen

Bevor mit den Vorarbeiten begonnen wird, müssen einige grundlegende Einstellungen vorgenommen werden, die einen erheblichen Einfluss auf die Qualität und Effizienz der optischen Zeichenerkennung haben. Diese Einstellungen beziehen sich auf die Bildauflösung, die Farbe des Bildes und das Dateiformat, in dem das Bild gespeichert wird.

4.2.1 Bildauflösung

Die Bildauflösung ist ein kritischer Faktor, der die Menge der Pixel in einem Bild bestimmt. Eine höhere Auflösung bedeutet mehr Pixel, was zu detaillierteren Bildern führt. Allerdings kann eine zu hohe Auflösung die Verarbeitungszeit erhöhen und mehr Speicherplatz benötigen. Daher ist es wichtig, ein Gleichgewicht zwischen der Detailgenauigkeit und der Verarbeitungseffizienz zu finden.

4.2.2 Farbe

Die Farbe des Bildes kann auch die Effizienz der optischen Zeichenerkennung beeinflussen. Bilder in Graustufen oder Schwarz-Weiß können einfacher zu verarbeiten sein als farbige Bilder, da sie weniger Farbdaten enthalten. Außerdem können bestimmte Farben die Erkennung von Textelementen erleichtern oder erschweren.

4.2.3 Dateiformat

Das Dateiformat, in dem das Bild gespeichert wird, kann ebenfalls einen Einfluss auf die optische Zeichenerkennung haben. Einige Formate, wie JPEG oder PNG, können Kompressionsartefakte enthalten, die die Texterkennung erschweren können. Andere Formate, wie TIFF oder BMP, können eine höhere Bildqualität bieten, benötigen aber mehr Speicherplatz.

4.3 Vorverarbeitung

Die Vorverarbeitung ist der Schritt vor der OCR und ein unerlässlicher Schritt, da sie die Qualität des zu analysierenden Bildes verbessert. Im Kontext dieser Arbeit umfasst die Vorverarbeitung das

Drehen und Ausschneiden des Bildes. Da der Mikrocontroller kopfüber am Zähler montiert ist, müssen die aufgenommenen Bilder um 180 Grad gedreht werden. Anschließend wird nur der relevante Teil des Bildes, der den Zählerstand enthält, ausgeschnitten, wie in Abbildung 2 dargestellt. Diese Schritte erleichtern es der OCR-Engine, die Bilder zu analysieren, da alle irrelevanten Texte dadurch entfernt werden.

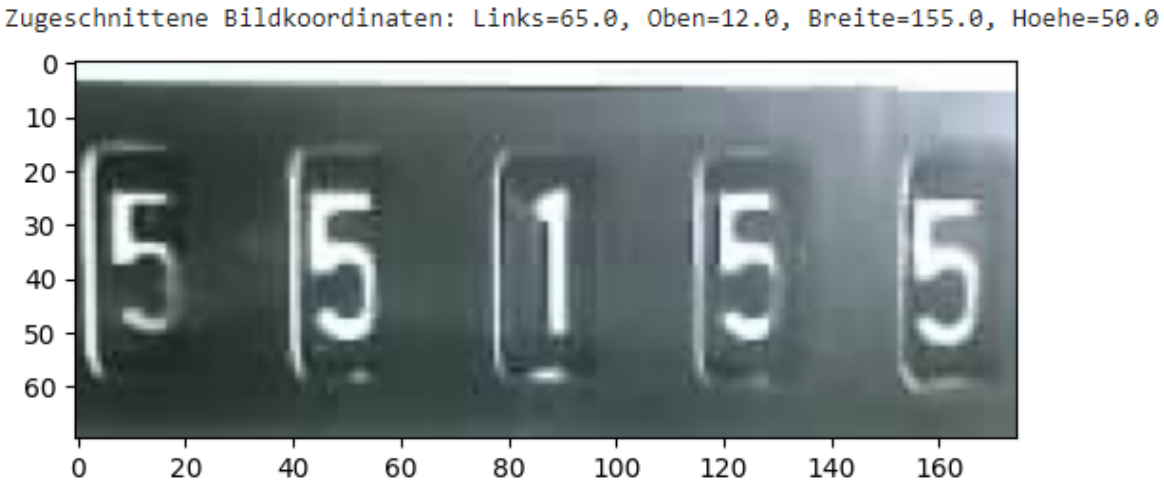


Abbildung 5 Bildausschnitt Vorverarbeitung mit Koordinaten und Rauschen

Abbildung 2 zeigt den Bildausschnitt vor der Vorverarbeitung mit den Koordinaten und dem vorhandenen Rauschen. Unter Rauschen versteht man unerwünschte oder störende Elemente in einem Bild, die die Qualität und Klarheit des Bildes beeinträchtigen können. In diesem Fall ist das Rauschen sichtbar als unerwünschte Pixel oder "Störungen" im Bild, die nicht zum eigentlichen Zählerstand gehören. Die Reduzierung oder Eliminierung dieses Rauschens ist ein weiterer wichtiger Schritt in der Vorverarbeitung, den die OCR-Engine zerlegt das Bild in seine einzelnen Pixel und betrachtet dieses binär. Jeder fehlerhafte Pixel reduziert die Genauigkeit der OCR-Engine.

Zur Bereinigung des Rauschens können schon einfache online Tools ausreichend sein [11]. Mit abgeschlossener Vorverarbeitung kann die Segmentierung begonnen werden.[12]

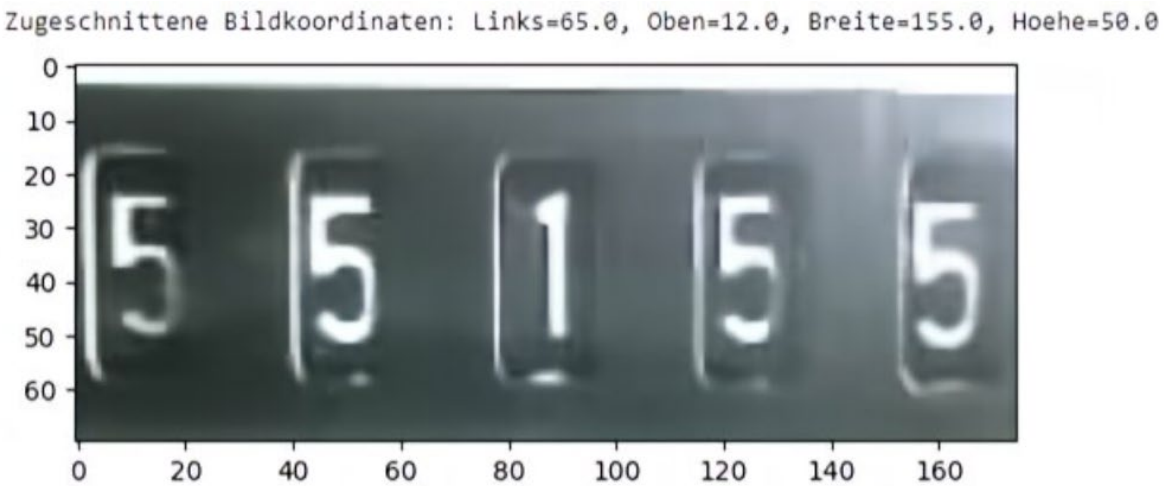


Abbildung 6 Bildausschnitt Vorverarbeitung bereinigt

4.3.1 Segmentierung:

Die Segmentierung ist der nächste Schritt in der Vorverarbeitung von Bildern. Sie dient dazu, die Inhalte des Bildes in verschiedene Segmente zu zerlegen, um die Analyse und Interpretation zu erleichtern. Im Allgemeinen kann die Segmentierung dazu verwendet werden, um Bild- und Textelemente zu trennen und die erkannten Textelemente in einzelne Zeichen zu zerlegen.

Ein wichtiger Aspekt der Segmentierung ist die Umwandlung des Bildes in ein Binärbild. Ein Binärbild besteht nur aus zwei Farben, in der Regel Schwarz und Weiß, die durch die Werte 0 und 1 repräsentiert werden. Dies vereinfacht die Interpretation des Bildes erheblich, da die Menge der zu analysierenden Daten reduziert wird. Abbildung 7

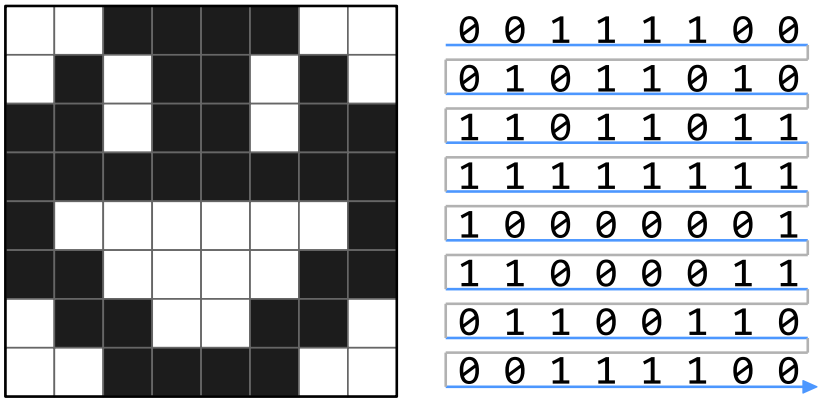


Abbildung 7 Beispiel Binärisierung [13]

Die Umwandlung in ein Binärbild erfolgt in der Regel durch die Anwendung eines Schwellenwertes. Ein Schwellenwert ist ein bestimmter Wert, der zur Unterscheidung zwischen den Pixeln des Bildes verwendet wird. Dabei wird ein Schwellenwert für die Helligkeit des Pixels festgelegt. Wenn der Pixel heller ist als der festgelegte Schwellenwert, wird er als weiß gewertet, ansonsten als schwarz.

Es ist wichtig zu beachten, dass die Wahl des Schwellenwertes einen erheblichen Einfluss auf das Ergebnis der Segmentierung hat. Ein zu hoher Schwellenwert kann dazu führen, dass wichtige Details verloren gehen, während ein zu niedriger Schwellenwert zu viel Rauschen im Bild erzeugen kann, ersichtlich in Abbildung 5.

Abbildung 8 Schwellenwert zu hoch und zu niedrig

In der Praxis kann die Segmentierung auch komplexere Verfahren umfassen, wie zum Beispiel die Erkennung von Kanten oder die Gruppierung von Pixeln basierend auf ihrer Farbe oder Textur. Diese

Verfahren können dazu beitragen, die Qualität der Segmentierung zu verbessern und die Genauigkeit der nachfolgenden Analyse zu erhöhen.

Es ist auch wichtig zu beachten, dass die Segmentierung nicht immer perfekt ist. Insbesondere bei komplexen Bildern oder Bildern mit schlechter Qualität kann es vorkommen, dass einige Segmente nicht korrekt erkannt werden oder dass einige Details verloren gehen, siehe Abbildung 9. In solchen Fällen kann es notwendig sein, zusätzliche Vorverarbeitungsschritte durchzuführen oder die Parameter der Segmentierung anzupassen. [14, 15]

Abbildung 9 fehlerhafte Segmentierung und optimale Segmentierung

4.3.2 Charakterisierung:

Die Charakterisierung ist der Schritt im OCR-Prozess, bei dem die Merkmale jedes einzelnen Zeichens analysiert und identifiziert werden. Diese Merkmale sind entscheidend für die spätere Klassifizierung und Erkennung der Zeichen. Im Folgenden werden einige der wichtigsten Techniken zur Charakterisierung in der OCR vorgestellt.

1. **Zoning:** Bei der Zoning-Methode wird das Bild eines Zeichens in mehrere Zonen unterteilt und aus jeder Zone werden Merkmale extrahiert. Dies kann dazu beitragen, Unterschiede in verschiedenen Teilen eines Zeichens zu identifizieren, was besonders nützlich ist, wenn zwei Zeichen insgesamt ähnlich sind, aber in bestimmten Bereichen Unterschiede aufweisen. Zum Beispiel könnte das Zeichen "8" in der oberen Zone eine kreisförmige Form haben, ähnlich wie das Zeichen "9", aber in der unteren Zone hat das Zeichen "8" eine zusätzliche kreisförmige Form, die das Zeichen "9" nicht hat. Durch die Zonierung können solche Unterschiede erkannt und zur Unterscheidung der Zeichen genutzt werden.

2. **Konturanalyse:** Die Konturen eines Zeichens können wertvolle Informationen über seine Form und Struktur liefern. Durch die Analyse der Konturen kann das OCR-System Unterschiede zwischen ähnlich aussehenden Zeichen erkennen. Zum Beispiel könnten die Zeichen "0" und "O" ähnlich aussehen, aber ihre Konturen könnten Unterschiede in der Rundheit oder Dicke aufweisen, die zur Unterscheidung der Zeichen genutzt werden können.

3. **Projektionsprofile:** Projektionsprofile sind Histogramme der Pixelintensitäten entlang der horizontalen oder vertikalen Achse des Bildes. Sie können verwendet werden, um die vertikale und horizontale Verteilung der Pixel in einem Zeichen zu analysieren. Diese Informationen können zur Unterscheidung von Zeichen mit ähnlicher Form, aber unterschiedlicher Pixelverteilung genutzt werden, wie zum Beispiel "m" und "n" oder "p" und "q".

4. **Crossing Counts:** Crossing Counts sind die Anzahl der Male, die eine Linie, die durch das Bild gezogen wird, die Kontur des Zeichens kreuzt. Sie können zur Unterscheidung von Zeichen mit ähnlicher Form, aber unterschiedlicher Anzahl von Linien oder Schleifen genutzt werden, wie zum Beispiel "8" und "9" oder "6" und "b".

5. **Geometrische Eigenschaften:** Geometrische Eigenschaften wie Fläche, Umfang, Kompaktheit, Orientierung, usw. können auch zur Charakterisierung von Zeichen genutzt werden. Sie können zur Unterscheidung von Zeichen mit ähnlicher Form, aber unterschiedlicher Größe, Dicke oder Orientierung genutzt werden.

Hierbei ist zu beachten, dass die Effektivität dieser Charakterisierungstechniken stark von der Qualität der Vorverarbeitung und Segmentierung abhängt. Eine gute Vorverarbeitung und Segmentierung können die Qualität der Charakterisierung und damit die Genauigkeit der gesamten OCR erheblich verbessern. [14, 15]

4.4 Klassifizierung und Erkennung

Nach abgeschlossener Vorarbeit werden Klassifizierung und Erkennung durchgeführt. Während andere Prozesse die Daten für die Analyse konditionieren, sind es die Klassifizierung und Erkennung, die den eigentlichen Kern der Zeicheninterpretation bilden. Dieses Kapitel widmet sich der detaillierten Untersuchung der zugrunde liegenden Mechanismen und Algorithmen dieser beiden Phasen und beleuchtet ihre entscheidende Rolle in der Transformation von visuellen Daten in strukturierte und interpretierbare Textinformationen.

4.4.1 Klassifizierung

Die Klassifizierung stellt einen essenziellen Schritt innerhalb des OCR-Verfahrens dar. In dieser Phase werden die während der Charakterisierung extrahierten Merkmale eines Zeichens herangezogen, um dieses einer spezifischen Kategorie oder Klasse zuzuordnen. Die Präzision dieses Schrittes beeinflusst maßgeblich die Identifikation des Zeichens in nachgelagerten Erkennungsprozessen.

Im Kontext der Klassifizierung werden die Merkmale des Zeichens, die in der vorherigen Charakterisierungsphase extrahiert wurden, systematisch analysiert. Dies geschieht durch einen Vergleich mit einer referenziellen Datenbank, die Merkmale bekannter Zeichen enthält. Das primäre Ziel dieses Prozesses ist die Identifikation des Zeichens oder zumindest die Eingrenzung potenzieller Kandidaten für das Zeichen.

Der k-NN (k-nearest neighbors) Algorithmus wird als exemplarisches Verfahren vorgestellt zur Klassifizierung. Der k-NN-Algorithmus repräsentiert ein klassisches Verfahren in der Mustererkennung und wird häufig für Klassifizierungsaufgaben herangezogen. Seine Funktionsweise basiert auf einem vergleichenden Ansatz, bei dem ein unbekanntes Zeichen mit einer Menge bekannter Zeichen kontrastiert wird.

1. **Distanzberechnung:** Für ein gegebenes Zeichen wird die Distanz zu jedem in der Datenbank hinterlegten Zeichen kalkuliert. Hierbei kann beispielsweise die euklidische Distanz als Metrik herangezogen werden.
2. **Selektion der k nächsten Referenzen:** Die "k" Zeichen, die die geringste Distanz zum unbekannten Zeichen aufweisen, werden selektiert.
3. **Entscheidungsfindung:** Das unbekannte Zeichen wird jener Klasse zugeordnet, die innerhalb der "k" selektierten Referenzen am häufigsten repräsentiert ist.

Ein hypothetisches Szenario könnte ein Zeichen sein, das morphologische Ähnlichkeiten mit den Buchstaben "A", "H" oder "K" aufweist. Der k-NN-Algorithmus würde dieses Zeichen mit der referenziellen Datenbank abgleichen. Sollte die Mehrheit der "k" ähnlichsten Referenzen den Buchstaben "A" repräsentieren, so wird das unbekannte Zeichen entsprechend als "A" klassifiziert. [16, 17]

4.4.2 Erkennung

Nachdem die Klassifizierung die potenziellen Kategorien oder Klassen für ein Zeichen identifiziert hat, folgt der Schritt der Erkennung. Hierbei wird das spezifische Zeichen endgültig bestimmt und in maschinenlesbaren Text umgewandelt. Dieser Prozess baut auf den Ergebnissen der Klassifizierung auf und nutzt zusätzliche Techniken und Algorithmen, um die Genauigkeit der Zeichenerkennung zu maximieren.

Die Erkennungsphase ist das Herzstück des OCR-Verfahrens. Während die Klassifizierung versucht, das Zeichen einer allgemeinen Kategorie zuzuordnen, zielt die Erkennung darauf ab, das genaue Zeichen oder Wort zu identifizieren. Dies wird durch den Abgleich mit einer umfangreichen Datenbank von Zeichen und Wörtern erreicht, die in verschiedenen Schriftarten, Größen und Stilen vorliegen können.

Die Ergebnisse der Klassifizierung, wie beispielsweise durch den k-NN-Algorithmus ermittelt, bieten eine Grundlage für die Erkennungsphase. Wenn beispielsweise die Klassifizierung ein Zeichen als potenziellen Buchstaben "A", "H" oder "K" identifiziert hat, wird die Erkennung spezifisch diese Kandidaten mit dem Originalbild vergleichen, um das genaue Zeichen zu bestimmen. Ist das Zeichen erkannt, wird es im Anschluss als Maschinentext ausgeben. [17]

4.5 OCR-Engines:

Im Folgenden wird auf drei OCR-Engines eingegangen und begründet, welche Engine für das Projekt verwendet wird. Dabei handelt es sich um die Engines Tesseract, AWS und die OCR.Space API.

4.5.1 Tesseract:

Tesseract stellt eine hoch entwickelte OCR-Engine dar, deren Ursprünge in den Forschungs- und Entwicklungsanstrengungen der 1980er Jahre liegen. Sie wurde initial von den Hewlett-Packard Labs konzipiert und später von Google akquiriert und substantiell optimiert. Seit ihrer Freigabe als Open-Source-Projekt durch Google im Jahr 2006 hat sich Tesseract zu einer der führenden OCR-Engines auf dem Markt etabliert.

Im Zuge der kontinuierlichen Fortschritte in der Computertechnologie hat Tesseract signifikante Weiterentwicklungen und Verbesserungen erfahren. Die Engine ist nun in der Lage, eine breite Palette von Texttypen und -formaten zu erkennen und nutzt dabei hoch entwickelte OCR-Algorithmen sowie Techniken des maschinellen Lernens, um die Präzision der Texterkennung zu steigern. Ein zentraler Aspekt von Tesseract ist die Tatsache, dass sie bereits vortrainiert ist und durch die Integration zusätzlicher Trainingsdaten weiter optimiert werden kann.

Trotz dieser beeindruckenden Fähigkeiten und Vorteile wurde Tesseract für das vorliegende Projekt nicht berücksichtigt. Der primäre Grund hierfür ist der hohe Anforderungs- und Ressourcenaufwand, der mit dem Training der Engine einhergeht. Zudem kann die technische Komplexität, die mit der Anpassung und Verbesserung von Tesseract verbunden ist, eine signifikante Herausforderung darstellen. Daher wurde eine Entscheidung zugunsten einer weniger komplexen und ressourcenintensiven Lösung getroffen.[9]

4.5.2 AWS (Amazon Web Services):

Amazon Web Services (AWS) ist eine Tochtergesellschaft von Amazon, die 2006 gegründet wurde. AWS bietet eine Vielzahl von Cloud-Computing-Services an, darunter Rechenleistung, Speicher und

Datenbanken, die es den Benutzern ermöglichen, Anwendungen in der Cloud auszuführen. AWS hat sich zu einem der führenden Anbieter von Cloud-Computing-Services entwickelt und bedient Kunden in verschiedenen Branchen und Regionen.

Eine der Services von AWS ist die OCR-Lösung, die es Entwicklern ermöglicht, Text aus Bildern und Dokumenten zu extrahieren. Dieser Dienst nutzt fortschrittliche maschinelle Lernalgorithmen für eine präzise Texterkennung und kann sowohl gedruckten als auch handschriftlichen Text in verschiedenen Sprachen erkennen. Trotz der vielen Vorteile, die AWS OCR bietet, einschließlich seiner Skalierbarkeit und hohen Verarbeitungskapazitäten, wurde es für dieses Projekt nicht gewählt, hauptsächlich aufgrund seiner Komplexität und potenziell Kosten. [18]

4.5.3 OCR.space API:

Die OCR.space API ist eine optische Zeichenerkennungslösung, die eine einfache Methode zur Analyse von Bildern und mehrseitigen PDF-Dokumenten bietet und die extrahierten Textergebnisse im JSON-Format zurückgibt. Diese API ist in drei Stufen verfügbar: kostenlos, PRO und PRO PDF. Für den Kontext dieser Arbeit wurde die kostenlose Version verwendet, die ein Limit von 500 Anfragen pro Tag pro IP-Adresse hat.

Die OCR.space API nutzt fortschrittliche OCR-Algorithmen und maschinelles Lernen, um eine genaue Texterkennung zu gewährleisten. Sie unterstützt eine Vielzahl von Dateiformaten und Sprachen und hat die Fähigkeit, sowohl gedruckten als auch handschriftlichen Text zu erkennen. Zudem bietet sie Funktionen zur Verarbeitung komplexer Dokumentlayouts und Tabellenstrukturen.

Ein bemerkenswerter Aspekt der OCR.space API ist ihre Fähigkeit, kontinuierlich zu lernen. Durch die Nutzung einer großen Anzahl von Benutzern und die Verarbeitung einer Vielzahl von Textarten und -formaten verbessert und passt sie ihre Texterkennungsfähigkeiten kontinuierlich an. Diese Eigenschaft trägt zur Verbesserung der Genauigkeit und Zuverlässigkeit der Texterkennung im Laufe der Zeit bei.

Die API bietet verschiedene OCR-Engines mit unterschiedlicher Verarbeitungslogik. Es wird empfohlen, alle zu testen und dann die Engine zu verwenden, die das beste OCR-Ergebnis liefert. Für diese Arbeit wurde die Engine 2 gewählt, da sie besonders gut auf die Erkennung von Zahlen in Bildern trainiert ist.

Die API gibt Ergebnisse im JSON-Format zurück. Das Ergebnis enthält in der Regel den Exitcode, Fehlerdetails (falls aufgetreten) und eine Reihe von analysierten Ergebnissen für die Bild- / PDF-Seiten. [19]

5. Die Hardware

In dieser Arbeit wurden drei verschiedene Mikrocontroller mit integrierter Kamera für die Aufnahme von Zählerstandsbildern verglichen. Zwei der getesteten Mikrocontroller basieren auf dem ESP32, während der dritte auf dem Sigmstar SSD 202D SoC basiert.

5.1.1 ESP32-EYE:

Espressif hat das ESP32-EYE entwickelt, einen Mikrocontroller, der auf dem ESP32-Chip basiert. Dieser Chip ist ein leistungsstarker 32-Bit-Dual-Core-Prozessor, der sowohl Wi-Fi als auch Bluetooth unterstützt und sich durch hohe Leistung und geringen Stromverbrauch auszeichnet, was ihn zu einer hervorragenden Wahl für Projekte im Bereich des Internets der Dinge (IoT) macht.

Eine besondere Eigenschaft des ESP32-EYE ist die integrierte 2-Megapixel-Kamera. Mit einer maximalen Auflösung von bis zu 1600x1200 Pixeln ermöglicht sie die Aufnahme von Bildern und Videos. Durch die integrierte Gesichtserkennungs- und -verfolgungsfunktion bietet der Mikrocontroller interessante Möglichkeiten für Projekte, die Bildverarbeitung erfordern.

Der Mikrocontroller ist zudem mit 8 MB RAM und 4 MB Flash-Speicher ausgestattet. Dieser Speicherplatz kann zum Speichern des Anwendungscodes, zum Zwischenspeichern von Daten und für viele andere Zwecke genutzt werden.

Zusätzliche Komponenten auf dem ESP32-EYE sind eine rote und eine weiße LED. Sie können zur Anzeige von Betriebszuständen, zur Fehleranzeige oder zur Beleuchtung verwendet werden. Das ESP32-Eye ist für ca. 10 € erhältlich. Abbildung 10 [20]

Abbildung 10 ESP32-EYE [21]

Allerdings erfüllt das ESP32-EYE nicht die Anforderungen, die für die Ausleuchtung des Zählers erforderlich sind, da die integrierten LEDs nicht ausreichend hell sind. Zudem verfügt das ESP32-EYE nicht über einen eingebauten Akku und muss daher über eine externe Stromquelle versorgt werden. Des Weiteren wird noch ein passendes Gehäuse benötigt, siehe auch Kapitel 6.1. [21]

5.1.2 M5Stack UnitV2:

Die M5Stack UnitV2, ausgestattet mit dem Sigmstar SSD 202D SoC, stellt auf den ersten Blick eine vielversprechende Hardware-Option für OCR-Anwendungen und Datenanalysen dar. Mit einem dual Core Cortex-A7 Prozessor, der eine Taktrate von 1,2 GHz aufweist, DDR3-Speicher und Nand-Flash-Speicher ausgestattet, bietet dieses Modul die erforderliche Rechenleistung für anspruchsvolle Anwendungen. Zusätzlich ermöglicht die integrierte 1080p-Kamera hochauflösende Bilder und Videos.

Die UnitV2 arbeitet mit einer von M5Stack angepassten Linux-Version, die die Nutzung von Jupyter Notebooks erlaubt - eine Open-Source-Webanwendung, die das Erstellen und Teilen von Dokumenten ermöglicht, die Live-Code, Gleichungen, Visualisierungen und erläuternden Text beinhalten. Das System erschien daher vielversprechend für die geplante Umsetzung, bei der eine OCR-Engine installiert werden sollte, um Zählerstände auszuwerten und die Ergebnisse anschließend in der Google Cloud zu analysieren. Die M5Stack UnitV2 ist für ca. 80 € erhältlich.

Trotz dieser vielversprechenden Eigenschaften und der anfänglichen Begeisterung für die Möglichkeiten, die das System bot, zeigte die praktische Umsetzung jedoch einige wesentliche Beschränkungen. Insbesondere erlaubt die angepasste Linux-Version von M5Stack die Installation einer OCR-Engine nicht, was das Gerät für das geplante Projekt unbrauchbar macht.

Darüber hinaus stellte sich heraus, dass die Steuerung des Grove Ports über das Jupyter Notebook nicht möglich war, was die Ansteuerung einer externen LED zur Ausleuchtung der Zähler verhinderte. Dieser Mangel an Flexibilität und Kontrolle über die Hardware-Komponenten stellte ein weiteres entscheidendes Hindernis dar.

Insgesamt zeigte die praktische Erprobung der M5Stack UnitV2, dass trotz der hochwertigen Hardware und der ansprechenden Funktionen die Systembeschränkungen die geplante Umsetzung der OCR-Anwendungen und Datenanalysen verhindern. Daher wurde eine alternative Hardwarelösung gesucht, die die notwendige Kompatibilität und Flexibilität bietet.[22]

Abbildung 11 M5Stack UnitV2 [22]

5.1.1 TimerCamX:

Die Timer Cam X stellt einen starken Mikrocontroller dar, der auf dem ESP32-Prozessor basiert, analog zum ESP-EYE, und mit einer 3-Megapixel-Kamera ausgerüstet ist. Diese hohe Bildqualität bildet eine solide Basis für effektive Texterkennung. Ein besonderer Vorzug der Timer Cam X ist die integrierte Batterie, welche einen unabhängigen Betrieb erlaubt, ohne die Notwendigkeit einer kontinuierlichen externen Stromversorgung. Diese Eigenschaft erweitert den Anwendungsbereich des Geräts beträchtlich und erlaubt flexible Einsatzmöglichkeiten.

Die Beleuchtung des Zählers ist ein wichtiger Aspekt für die korrekte Erfassung der Zählerstände. In diesem Bereich punktet die Timer Cam X durch ihre Fähigkeit, eine externe LED über den Grove-Stecker zu integrieren und zu steuern. Diese Option erlaubt die optimale Anpassung der Beleuchtungsverhältnisse und sichert so eine hochqualitative Texterkennung.

Die Timer Cam X bietet zudem eine LEGO-Halterung an, die eine einfache und flexible Anbringung am Zähler ermöglicht. Diese Montagemöglichkeit erleichtert den Umgang mit der Kamera und ermöglicht eine präzise Positionierung des Geräts. Die Timer Cam X ist für ca. 20 € erhältlich.

Unter Berücksichtigung dieser Faktoren hat sich die Timer Cam X als ideale Wahl herausgestellt. Ihre Eigenschaften wie der unabhängige Betrieb, die individuell anpassbare Beleuchtung und die benutzerfreundliche Montage sind in ihrer Kombination einzigartig. Darüber hinaus bietet die robuste und flexible Bauweise der Timer Cam X eine hohe Zuverlässigkeit und Varianz in den Anwendungsmöglichkeiten. Aus diesen Gründen wurde die Timer Cam X als präferierte Lösung für dieses Projekt ausgewählt.

Zum Abschluss des Kapitels gibt es Tabelle 1 eine Zusammenfassung der Hardware. Gefolgt von einem UML-Diagramm zu Hardware, Software und Informationsfluss in Abbildung 13. [23]

Abbildung 12 Timer Cam X [23]

Microcontroller	Prozessor	Speicher	WiFi	Kamera	Akku	LED integriert	Groove Port für externe LED	Besonderheiten
ESP32-EYE	32-Bit-Dual-Core-Prozessor	8 MB RAM und 4 MB Flash-Speicher	Ja	2 Megapixel	Nein	ja	nein	keine
M5Stack Unit V2	Sigmstar SSD202D	128MB-DDR3 memory, 512MB NAND Flash	Ja	2 Megapixel	Ja	Nein	Ja	Eigne Linux Version von M5Stack, jedoch nicht nutzbar für Projekt. Da Hersteller eigene Version mit Limitierungen
Timer Cam X	32-Bit-Dual-Core-Prozessor	8 MB RAM und 4 MB Flash-Speicher	Ja	3 Megapixel	Ja	nein	Ja	keine

Tabelle 1 Vergleichstabelle Microcontroller

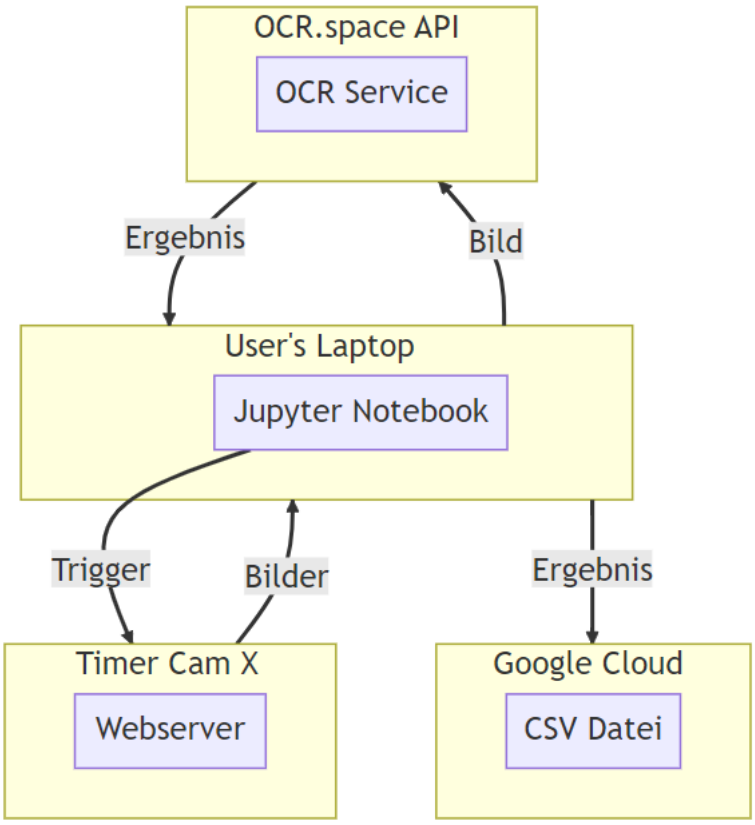


Abbildung 13 Aufbau als UML Verteilungsdiagramm

6. Umsetzung

In diesem Kapitel geht es um die praktische Umsetzung des Projekts. Zuerst wird der Versuchsaufbau vorgestellt, der als Ausgangspunkt diente. Danach wird der endgültige Aufbau am Zählerschrank beschrieben, der das fertige System repräsentiert. Abschließend gibt es eine Analyse der Ergebnisse, die zeigt, wie gut das System funktioniert.

6.1 3D-Gehäuse für das ESP32-EYE

Die Vorlage für das 3D-Gehäuse, speziell konzipiert für das ESP32-EYE, wurde kostenfrei von der Webseite www.thingiverse.com heruntergeladen und anschließend an der Hochschule Pforzheim gedruckt. Dieses Gehäuse war ursprünglich dafür vorgesehen, am Zähler befestigt zu werden. Im Laufe des Projekts stellte sich jedoch heraus, dass die Dimensionen des Gehäuses zu groß waren, was dazu führte, dass die Tür des Zählerschranks nicht geschlossen werden konnte. Darüber hinaus wurde festgestellt, dass die Timer Cam X nicht auf das Gehäuse passt. Die Timer Cam X verfügt über ein eigenes Gehäuse und kann mit LEGO Bauteilen montiert werden. Dadurch kann der Aufbau flexibler gestaltet werden und die externe LED kann besser Positioniert werden. Dadurch kann den Reflexionen besser entgegengewirkt werden. Aufgrund dieser Erkenntnisse wurde die Idee, das ESP Eye in einem Gehäuse zu verwenden, verworfen. Zukünftige Anpassungen und Überlegungen müssen diese Faktoren berücksichtigen, um sicherzustellen, dass das Gehäuse sowohl funktional als auch passend ist.

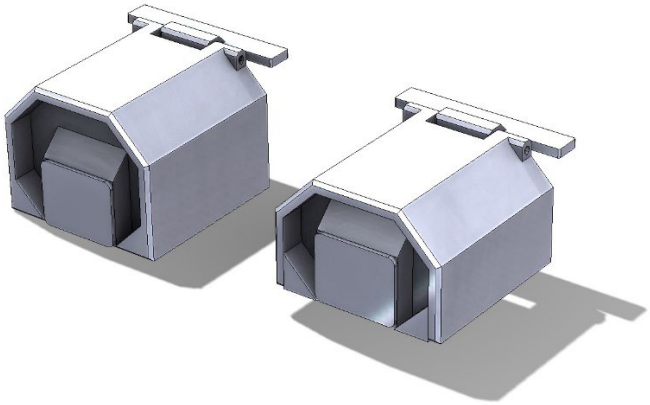


Abbildung 14 Skizze 3D Druck

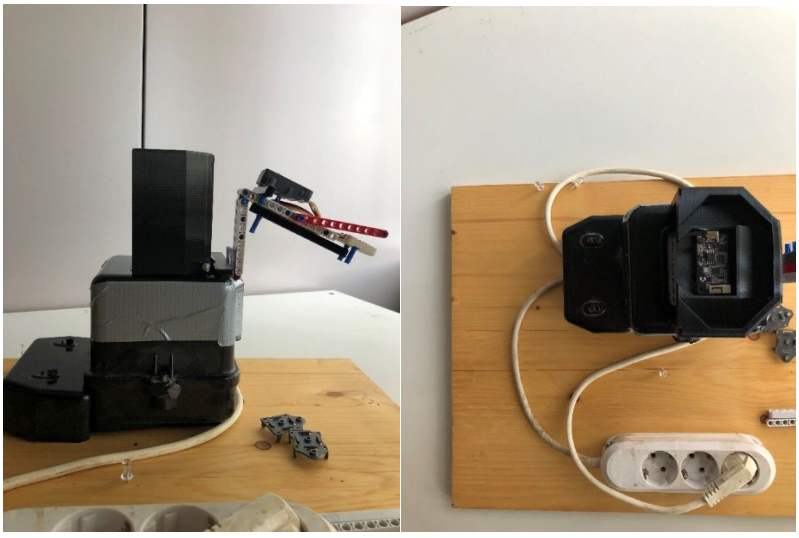


Abbildung 15 3D Gehäuse im Einsatz am Versuchsaufbau

6.2 Versuchsaufbau

Für den Versuchsaufbau wurde ein Baustellenwechselstromzähler verwendet, der auf eine Massivholzplatte montiert und mit einem Verlängerungskabel verbunden wurde. Die Wahl dieses speziellen Zählers wurde getroffen, um die Laufstrecken im Projekt zu reduzieren und da der Zählerschrank an der Hochschule nur mit Fachpersonal zugänglich war.

Um die Lichtverhältnisse im späteren Zählerschrank zu simulieren, wurde ein Karton in der Größe des Brettes zur Abdunkelung des Zählers verwendet. Diese Simulation war wichtig, um die Reflexion der LED im Dunkeln zu berücksichtigen, da sie sich von der Reflexion im Hellen unterscheidet. Durch die richtige Positionierung der LED konnten starke Reflexionen vermieden werden, die die Bildqualität verschlechtern und es der OCR-Engine erschweren könnten, etwas zu erkennen. [17]

An den Wechselstromzähler wurde mit Klebeband eine LEGO-Halterung montiert, um die TimerCamX X über dem Zählerstand zu platzieren. Die TimerCamX X wurde dann mithilfe eines Webservers justiert. Über den Webserver konnte ein Livestream abgerufen werden, um die richtige Position zu ermitteln und sicherzustellen, dass nur der Zählerstand und keine weiteren Texte abfotografiert wurden. Dies war notwendig, um die Menge an unnötigen Texten, wie z.B. Informationen über den Hersteller des Zählers, zu reduzieren, die sonst herausgefiltert werden müssten.

Die TimerCamX X wurde mit dem Standard-Webserver von Espressif geflasht. Im C++-Code wurde zusätzlich angepasst, dass die LED über den Grove-Port aktiviert wird, wenn der Capture-Befehl aufgerufen wird. Dieser Befehl ist der Standardbefehl zum Aufnehmen eines Bildes über den Webserver. Diese Anpassung war notwendig, um sicherzustellen, dass die LED nur dann aktiviert wird, wenn tatsächlich ein Bild aufgenommen wird, um unnötigen Stromverbrauch zu vermeiden und die Lebensdauer der LED zu verlängern. Nachdem der gesamte Aufbau eingestellt war, begannen die Arbeiten am Code in Python. Dabei wurde ein Jupyter Notebook verwendet. Dieses interaktive Tool ermöglichte es, den Code schrittweise zu entwickeln und sofortige Rückmeldungen zu den Ergebnissen zu erhalten. Es war besonders nützlich für die explorative Datenanalyse und die Visualisierung der Ergebnisse. Abbildung 16

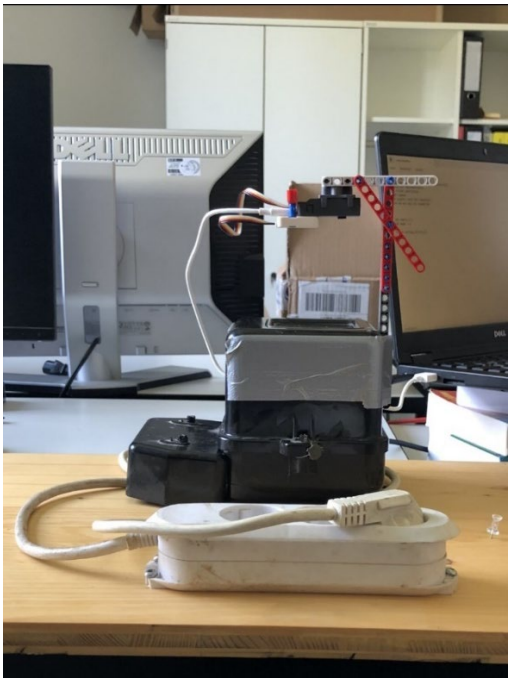


Abbildung 16 Versuchsaufbau TimerCamX Lego-Halterung

6.3 Schnittstellen verknüpfen

Um den Zählerstand als Maschinentext zu erhalten, wurden mehrere Schnittstellen miteinander verknüpft. Dieser Prozess beginnt mit der Nutzung eines Jupyter Notebooks, das den Webserver des Mikrocontrollers mit dem "/capture"-Befehl aufruft, um ein Bild zu erstellen und die LED des Mikrocontrollers zu aktivieren. Der Mikrocontroller, nimmt ein Bild des Zählerstands auf. Dieses Bild wird dann , zunächst vertikal um 180 Grad gedreht und anschließend wird der Zählerstand ausgeschnitten.

Anschließend wird das zu analysierende Bild an die API von OCR.Space gesendet, wo es ausgewertet wird. Das Ergebnis kommt im JSON-Format zurück. Aus der API-Antwort extrahiert der Code im Jupyter Notebook die Zahlen und überprüft diese zunächst auf ihre Vollständigkeit. Sollten die Zahlen nicht vollständig sein, kommt die Prüflogik zum Einsatz. Diese schaut sich jede einzelne Zahl der Antwort und des vorherigen Ergebnisses an, um die richtige fehlende Zahl zu ermitteln. Sollten es zu viele Zahlen sein, kommt die Segmentierungsfunktion zum Einsatz, welche Segmente erstellt und das am besten passende Segment für die weitere Plausibilitätsprüfung benutzt.

Sobald die OCR-Analyse und die Plausibilitätsprüfung abgeschlossen sind, wird das Ergebnis in einer CSV-Tabelle gespeichert. Diese Tabelle dient als Datenspeicher und kann für spätere Analysen und Berichte genutzt werden. Schließlich wird die CSV-Tabelle in Google OneDrive gespeichert. Dies ermöglicht einen einfachen Zugriff auf die Daten und ihre sichere Speicherung, siehe auch Abbildung 17.

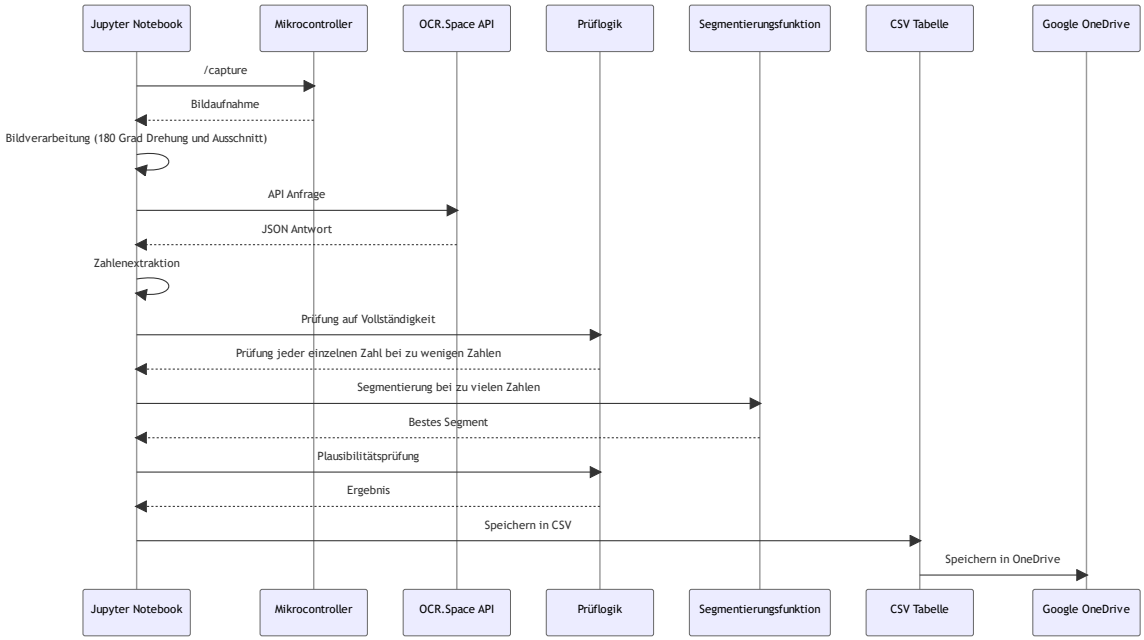


Abbildung 17 Systemablauf

Der gesamte Prozess wird durch einen Python-Code im Jupyter Notebook gesteuert, der die verschiedenen Schritte koordiniert und automatisiert. Dieser Code verwendet verschiedene Bibliotheken und Tools, darunter Requests für HTTP-Anfragen, PIL für die Bildverarbeitung, pandas für die Datenverarbeitung und Matplotlib für die Bildanzeige. Es ist wichtig zu beachten, dass während des Versuchsaufbaus kein fester Wert für die Plausibilitätsprüfung festgelegt wurde. Dieser Wert wurde später im finalen Aufbau getestet und festgelegt. Durch die Verknüpfung dieser verschiedenen Schnittstellen können wir den Zählerstand als Maschinentext erfassen und speichern.

6.4 Auswertung Versuchsaufbau

Die Auswertung des Versuchsaufbaus hat gezeigt, dass die Implementierung erfolgreich war und konstant plausible Zählerstände lieferte. Um den Zähler zum Laufen zu bringen, wurden Geräte wie ein Heißluftföhn mit einer Leistung von 2,7 kW verwendet. Dies ermöglichte es, den Zählerstand in Echtzeit zu erfassen und zu analysieren.

Ein Großteil der Tests konzentrierte sich auf das Abfangen des Fehlers, wenn der Zählerstand zwischen zwei Zahlen steht. Dies ist eine häufige Herausforderung bei der optischen Zeichenerkennung, da die Maschine Schwierigkeiten haben kann, unvollständige oder überlappende Zahlen zu erkennen. Um dieses Problem zu lösen, wurde eine spezielle Prüflogik implementiert. Diese Logik prüft die vorherigen Werte und gleicht jede einzelne Zahl des neuen Werts mit den alten ab. Wenn die Zahlen nicht ähnlich zueinander waren, hat der Code dies korrigiert oder die Zahlen verworfen.

Zusätzlich zur Prüflogik wurde eine Plausibilitätsprüfung durchgeführt. Diese Prüfung verhindert, dass Werte, die kleiner sind als die vorherigen, aufgenommen werden oder Werte, die größer sind als die maximale Schrittweite. Dies stellt sicher, dass keine fehlerhaften Werte gespeichert werden und erhöht die Genauigkeit der erfassten Zählerstände.

Darüber hinaus wurde durch die richtige Einstellung und Positionierung der LED die Reflexion so weit reduziert, dass der Versuchsaufbau konstant plausible Zahlenwerte lieferte. Die Reflektion kann das

Bild verzerren und die Genauigkeit der optischen Zeichenerkennung beeinträchtigen. Daher war es wichtig, die Lichtverhältnisse zu optimieren und die Reflektion zu minimieren.

Die größte Herausforderung lag bei der Implementierung der Prüflogik und der Bewältigung der Reflektionen. Beide Aspekte erforderten eine sorgfältige Planung und Anpassung, um sicherzustellen, dass der Versuchsaufbau korrekte Ergebnisse liefert. Trotz dieser Herausforderungen war der Versuchsaufbau insgesamt erfolgreich und konnte den Zählerstand effizient und genau als Maschinentext erfassen.

Der Versuchsaufbau hat eine solide Grundlage für den endgültigen Aufbau geschaffen. Mit den gewonnenen Erkenntnissen und dem erstellten Prototyp konnte nun der finale Aufbau angegangen werden.

6.5 Finaler Aufbau

Der finale Aufbau des Projekts, obwohl er in seiner Struktur dem Versuchsaufbau ähnelt, beinhaltete spezifische Anpassungen und Verbesserungen, die auf den während der Testphase gewonnenen Erkenntnissen basierten.

Eine Änderung im finalen Aufbau war die Neupositionierung der LED. Aufgrund der spezifischen Lichtverhältnisse vor Ort musste die LED neu positioniert werden, um die Reflektionen zu minimieren und den Zählerstand klar und deutlich sichtbar zu machen. Die richtige Positionierung und Einstellung der LED waren entscheidend für die Qualität der erfassten Bilder und damit für die Genauigkeit der Zählerstandserfassung.

Zusätzlich zur Neupositionierung der LED musste auch der Bildausschnitt neu angepasst werden. Dies war notwendig, um sicherzustellen, dass der relevante Teil des Zählers vollständig und korrekt im Bild erfasst wurde. Trotz dieser Änderungen blieb der Rest des Aufbaus im Wesentlichen unverändert, basierend auf dem ursprünglichen Versuchsaufbau.

Um eine zuverlässige und stabile Verbindung für den Mikrocontroller zu gewährleisten, wurde ein zusätzlicher WLAN-Punkt in der Nähe des Zählers freigeschaltet. Dies war besonders wichtig, da der Zähler der Hochschulbibliothek verwendet wurde und sich dieser im Keller befindet. Durch die Einrichtung eines zusätzlichen WLAN-Punktes konnte eine stabile und störungsfreie Übertragung sichergestellt werden.

Zu Beginn des Projekts wurde das Programm manuell ausgelöst. Dies ermöglichte es, ein Gefühl für die Zählergeschwindigkeit zu entwickeln und die Schrittweite der Plausibilitätsprüfung des Codes richtig einzustellen. Dieser manuelle Prozess wurde so lange fortgesetzt, bis die Reflektionen minimiert waren und die Ergebnisse zufriedenstellend waren. Nach mehreren Tests wurde eine maximale Schrittweite von 3 kWh als optimal ermittelt.

Im späteren Verlauf des Projekts wurde das Programm automatisiert und mithilfe des Windows-Aufgabenplaners alle 15 Minuten ausgeführt. Dafür wurde eine Batch-Datei erstellt, die das Python-Programm aufruft. Die Aufgabenplanung wurde auch so eingerichtet, dass Sie sofort startet, wenn der Computer gestartet wird. Diese Automatisierung ermöglichte eine kontinuierliche und effiziente Erfassung des Zählerstands und minimierte die Notwendigkeit manueller Eingriffe.

7. Ergebnis

Im Kapitel "Ergebnis" werden zwei Hauptaspekte behandelt: Erstens die Genauigkeit und Zuverlässigkeit des OCR-basierten Stromzählers und zweitens die Analyse des Stromverbrauchs der Hochschulbibliothek.

7.1 Genauigkeit des OCR-Zählers

Der Stromzähler mit optischer Zeichenerkennung (OCR) hat über einen Zeitraum von zwei Wochen Daten gesammelt, die später in einer CSV-Datei ausgewertet wurden. Als ersten Schritt haben wir ein Diagramm erstellt, in das sowohl eine Trendlinie als auch die individuellen Messpunkte eingefügt wurden. Die Trendlinie stellt den errechneten Durchschnittsverbrauch über diese zwei Wochen dar. Wie aus Abbildung 18 ersichtlich, liegen alle Messwerte nahe der Trendlinie, was auf eine ordnungsgemäße Funktion des Zählers schließen lässt.

Am Anfang der Messreihe ist eine Abweichung von der Trendlinie zu beobachten, die durch eine manuelle Auslösung verursacht wurde. Danach wurde der Windows-Aufgabenplaner verwendet, um in 15-Minuten-Intervallen Messungen vorzunehmen. Ab diesem Zeitpunkt wurden die Daten kontinuierlich erfasst und lagen stets tagesweise nahe der Trendlinie. Dies bestätigt, dass die Plausibilitätsprüfung korrekt arbeitet, da alle Punkte entweder gleichbleibend oder ansteigend sind.

Das Diagramm zeigt auch, dass die Abstände zwischen den einzelnen Messpunkten nicht gleichmäßig sind. Dies lässt sich auf zwei Faktoren zurückführen. Erstens hat die Prüflogik gelegentlich eine falsche Zahl ausgewählt, was zu Sprüngen in den Daten führte. Nach diesen Sprüngen wurden keine weiteren Punkte hinzugefügt, bis der Sprungwert wieder erreicht war. Zweitens hat die OCR-Engine in einigen Fällen Zahlen fehlerhaft interpretiert, wie z.B. bei den Zahlen 3 und 5. Wenn die Zahl 3 nur teilweise sichtbar ist, kann sie aufgrund ihrer Form von der OCR-Engine fälschlicherweise als 5 interpretiert werden. Diese Art von Fehlern kann durch die eingestellte Prüflogik und Plausibilitätsprüfung nicht erkannt werden, da sie innerhalb des festgelegten Intervalls liegen.

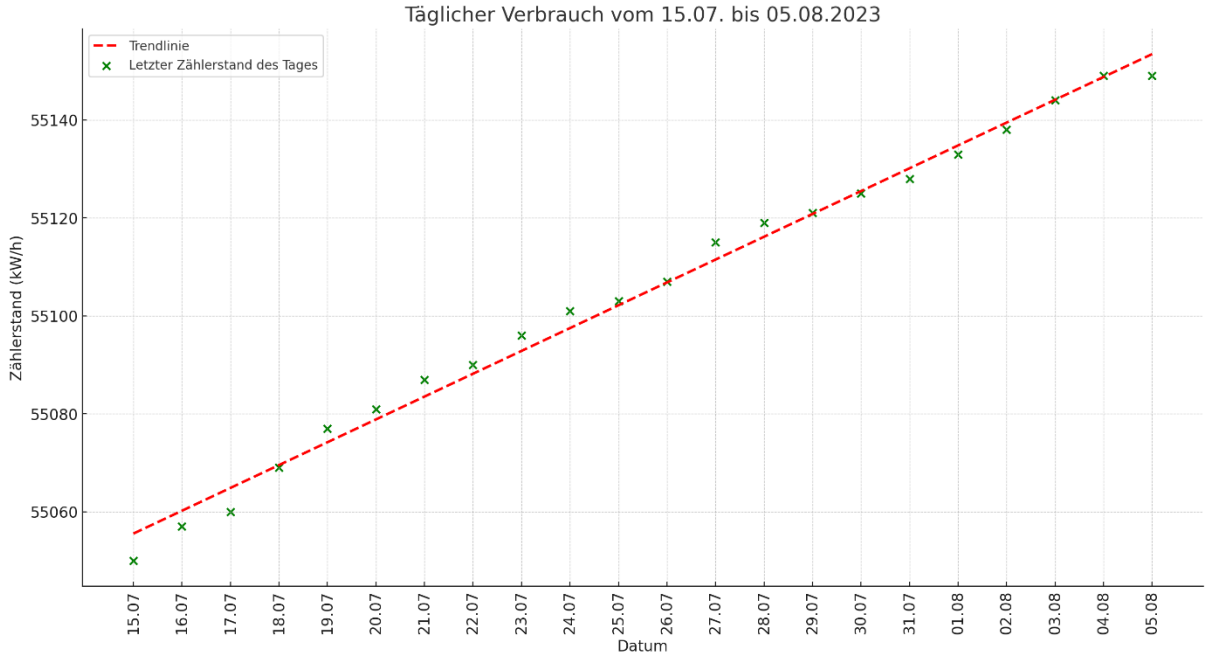


Abbildung 18 Auswertung Zählerstand mit Trendlinie von 13.07.23-05.08.23[2]

7.2 Analyse des Stromverbrauchs

Um einen detaillierten Einblick in den Stromverbrauch der Hochschulbibliothek zu erhalten, wird zunächst der Tagesverlauf des 14.07.2023 betrachtet, der Tag mit den meisten korrekten Zählerständen.

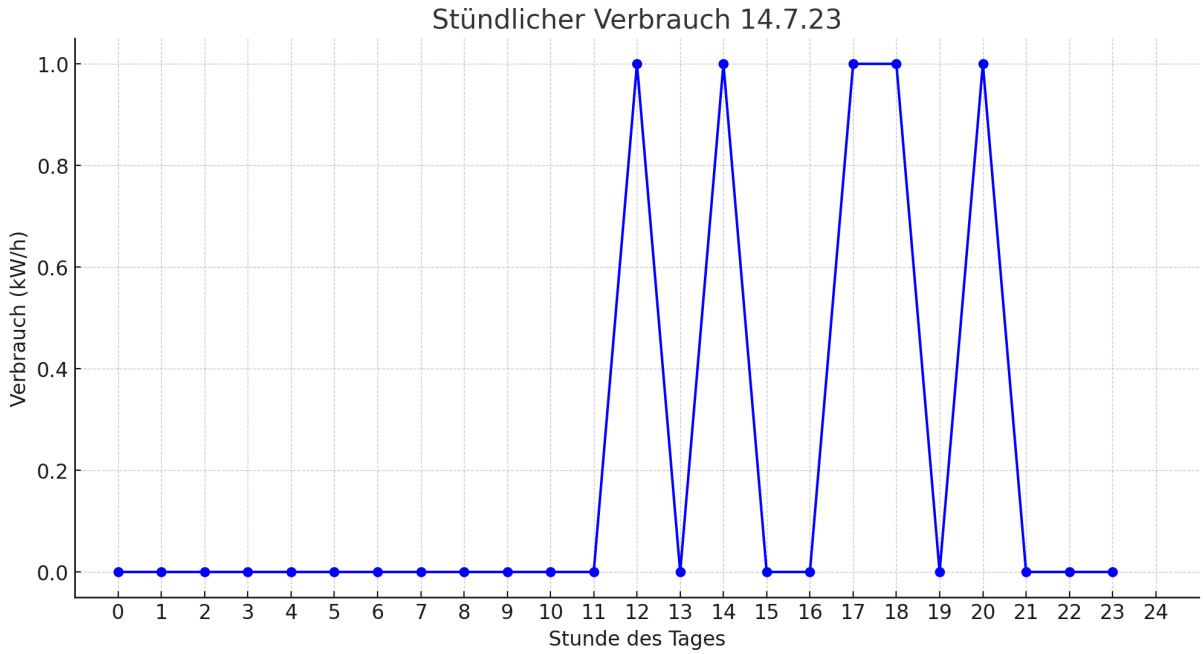


Abbildung 19 Stündlicher Stromverbrauch 14.07.2023

Abbildung 19 zeigt das die meisten Verbräuche zwischen 11 und 21 Uhr stattgefunden haben. Der Verbrauch ist relativ gering mit einer maximalen Rate von 1 kW/h.

Abschließend werfen wir einen Blick auf Abbildung 20, die den täglichen Verbrauch über einen Zeitraum von zwei Wochen in Form eines Säulendiagramms darstellt. Dieses Diagramm beinhaltet ebenfalls eine Trendlinie, die den durchschnittlichen Stromverbrauch während dieser zwei Wochen repräsentiert. Abbildung 20 bestätigt den in Abbildung 19 beobachteten Trend: Der Stromverbrauch bleibt relativ konstant und niedrig.

Insgesamt ist der Stromverbrauch für eine Bibliothek sehr gering. Dies kann darauf zurückgeführt werden, dass nur einer von drei Zählern betrachtet wurde. Für diese Analyse wurde bewusst der Zähler mit der langsamsten Laufgeschwindigkeit ausgewählt, um den gesamten Zählerstand über einen möglichst langen Zeitraum sichtbar zu machen und häufig fotografieren zu können.

Wenn die Zählerstände tagesweise betrachtet werden, zeigt das System eine bessere Leistung. Denn es gibt noch Schwächen bei der stündlichen Betrachtung, wie die begrenzte Anzahl an Messpunkten und die gelegentliche Nichterkennung von Zählerständen zeigen.

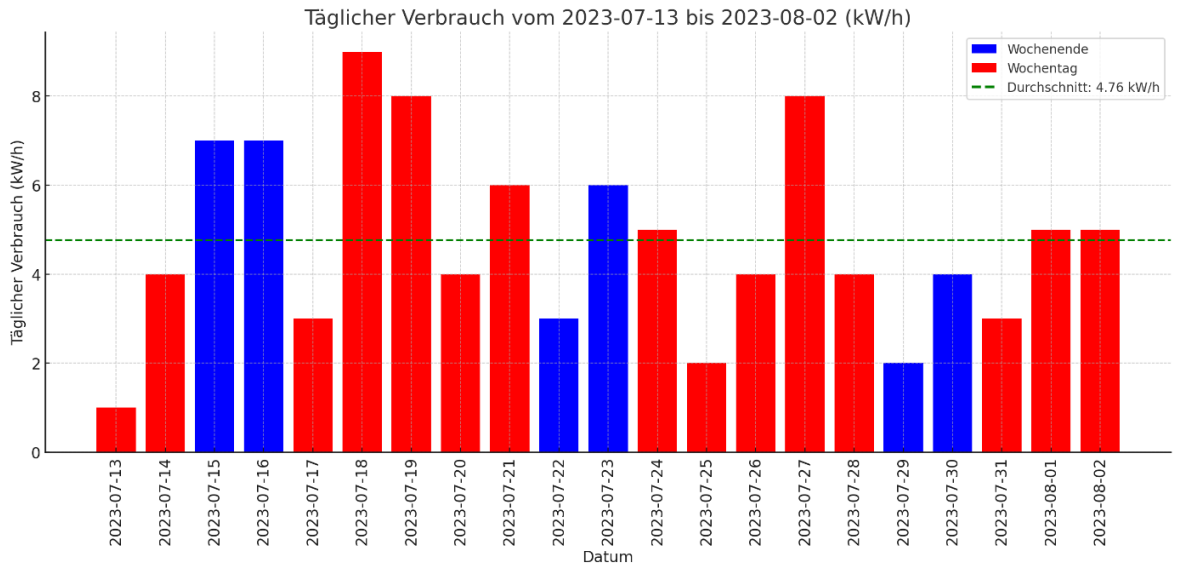


Abbildung 20 Verbrauch Zeitraum 13.07-05.08.23

8. Fazit

Das Projekt hat gezeigt das es Machbar ist, mit einfachen Mitteln ein Prototyp mit schwächen zu entwickeln. Der Prototyp hat es ermöglicht erste Daten, digital zum Stromverbrauch der Hoschulbibliothek zu sammeln. Dadurch kamen Rückfragen auf die zu der Erkenntnis führten, dass zum Beispiel die Steuerungseinheit für das Licht defekt ist. Dadurch kann das Licht manchmal nicht abgeschaltet werden und läuft die ganze Nacht durch. Eine Reparatur würde helfen weiter Energie zu sparen. Weiterführend könnten noch mehr Stromzähler in das System aufgenommen werden. Um den Gesamtverbrauch besser zu analysieren, da die Bibliothek über drei Stromzähler verfügt. Leider war es nicht möglich heraus zu finden, für welchen Stromkreislauf der genutzte Stromzähler war.

Die entdeckten Schwächen zeigen Herausforderungen und zu verbessernde Aspekte auf, wie folgend:

1. **LED-Dimmung und Bildqualität:** In der aktuellen Implementierung lief die LED ständig mit voller Leistung, was die korrekte Positionierung erschwerte und die vollständige Eliminierung von Reflektionen verhinderte. Die führte zu einer leicht verminderten Bildqualität. Aufgrund fehlender Kenntnisse in der C++-Programmierung konnte keine dimmbare LED implementiert werden.

2. **Einsatz von Maschinellem Lernen:** Das System hätte von der Integration maschinellen Lernens profitieren können, insbesondere bei der Plausibilitätsprüfung der erfassten Zählerstände. Derzeit muss der Benutzer manuelle Korrekturen vornehmen, wenn über längere Zeit kein Zählerstand erkannt wird. Maschine Learning könnte auch für die Bildvorverarbeitung genutzt werden.

3. **Remote-Computing und Systemstabilität:** Aufgrund eines Remote-Verbots im Arbeitsumfeld konnte das Programm nicht auf einem Remote-Computer implementiert werden. Dies führte immer wieder zu Störungen, wie etwa durch Updates jeglicher Art.

4. **Zugang zu Zählerschränken und Testbedingungen:** Der Zugang zu den Zählerschränken war eingeschränkt und erforderte die ständige Anwesenheit von Personal. Dadurch war die Test Zeit am Zähler relativ gering.

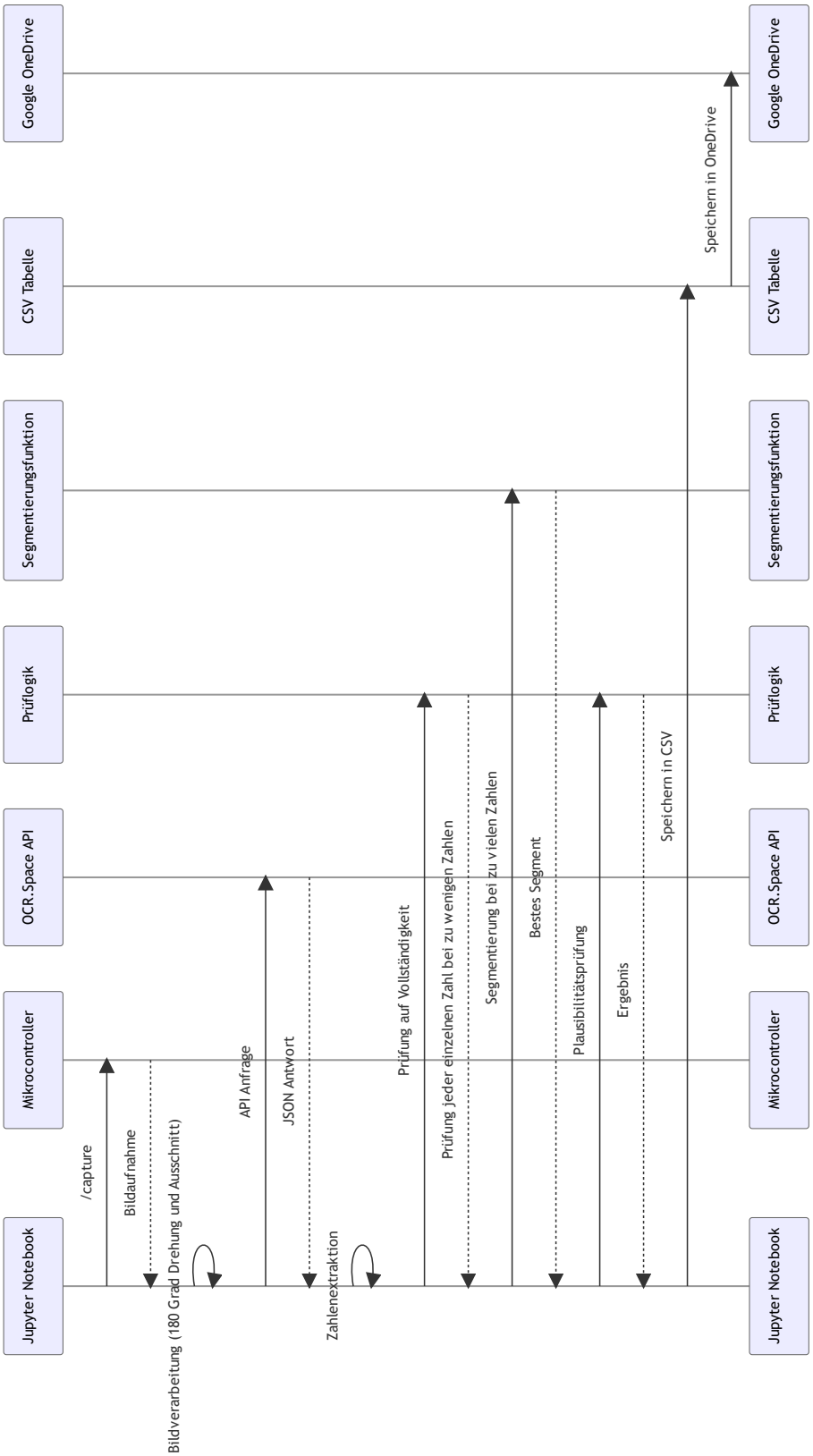
5. **Hardware-Implementierung und Mikrocontroller:** Die Auswahl und Konfiguration der Mikrocontroller stellten sich als komplexer heraus als anfangs angenommen. Vor allem der Umgang mit der M5Stack Unit V2 und ihrem Linux Betriebssystem. Es wurde vergebens versucht aktuelle Python Bibliotheken auf dem Jupyter Notebook zu installieren. Hier stellt sich heraus, dass es eine eigene Linuxversion des Herstellers ist, die dies nicht erlaubt. Aufgrund der mangelnden Erfahrung in diesem Bereich, verzögerten solche Erfahrungen den Fortschritt stetig.

6. **Langzeittests und Verbrauchsanalyse:** Der Testzeitraum war auf zwei Wochen während der Semesterferien begrenzt. Eine Analyse der Verbrauchswerte während der Vorlesungszeit im Vergleich zur vorlesungsfreien Zeit wurde nicht durchgeführt.

7. **Webinterface:** Es könnte noch ein Webinterface integriert werden um die Messdaten, grafisch besser aufzubereiten.

Abschließend lässt sich sagen der Prototyp wurde erstellt, getestet und hat Daten geliefert. Mit Hilfe der Daten konnten Optimierungsaspekte für den OCR-Zähler und auch für den Stromverbrauch der Hoschulbibiliothek aufgezeigt werden. Die Verbesserungen könnten in zukünftige Projekte weiter eingearbeitet werden.

9. Anhang A


Google OneDrive
CSV Tabelle
Segmentierungsfunktion
Prüflogik
OCR.Space API
Mikrocontroller
Jupyter Notebook
/capture
Bildaufnahme
Bildverarbeitung (180 Grad Drehung und Ausschnitt)
API Anfrage
JSON Antwort
Zahlenextraktion
Prüfung auf Vollständigkeit
Prüfung jeder einzelnen Zahl bei zu wenigen Zahlen
Segmentierung bei zu vielen Zahlen
Bestes Segment
Plausibilitätsprüfung
Ergebnis
Speichern in CSV
Speichern in OneDrive
Google OneDrive
CSV Tabelle
Segmentierungsfunktion
Prüflogik
OCR.Space API
Mikrocontroller
Jupyter Notebook

10. Anhang B

```
# Importieren der benötigten Bibliotheken
import requests
from PIL import Image
from io import BytesIO
import pandas as pd
from datetime import datetime
import matplotlib.pyplot as plt
import time


# Definieren der Konstanten
BILD_URL = ""  # URL des zu analysierenden Bildes
API_URL = "https://api.ocr.space/parse/image"  # URL der OCR-API
API_KEY = ""  # API-Schlüssel
SPRACHE = "eng"  # Sprache der Zeichen auf dem Bild
ROTATIONS_GRAD = 180  # Gradzahl der Rotation, die auf das Bild ange-
wendet wird
MAX_DIFFERENZ = 3  # Maximal zulässiger Unterschied zwischen neuen und
alten Zählerwerten
ZAEHLER_DATEN_DATEI = r"G:\\\\Meine Ablage\\\\Colab Notebooks\\\\zah-
lerdaten.xlsx"  # Pfad zur Datei, in der die Zählerdaten gespeichert
werden
# Setzen Sie den Timer auf 15 Minuten (15 Minuten * 60 Sekunden/Minute)
TIMER_INTERVAL = 15 * 60

# Funktion zum Herunterladen, Drehen und Zuschneiden des Bildes
def bild_herunterladen_drehen_und_zuschneiden(bild_url, rotations_grad,
links, oben, breite, hoehe):
    # Bild von der URL herunterladen
    antwort = requests.get(bild_url)
    img = Image.open(BytesIO(antwort.content))

    # Bild drehen
    img = img.rotate(rotations_grad)

    # Bild zuschneiden (um 10 in alle Richtungen erweitert, um sicher-
zustellen, dass der gesamte Zählerstand im Bild ist)
    img = img.crop((links-10, oben-10, links+breite+10, oben+hoehe+10))

    # Bild in JPEG-Format konvertieren
    with BytesIO() as ausgabe:
        img.save(ausgabe, format="JPEG")
        jpg_daten = ausgabe.getvalue()

    return jpg_daten, img
```

```
# Funktion zum Senden der API-Anfrage
def api_anfrage_senden(api_url, api_key, sprache, jpg_daten):
    # Parameter für die API-Anfrage festlegen
    params = {
        "apikey": api_key,
        "language": sprache,
        "OCREngine": 2  # Verwendung von OCR Engine 2
    }

    # API-Anfrage senden und Antwort zurückgeben
    antwort = requests.post(api_url, files={"image.jpg": jpg_daten},
data=params)
    return antwort.json()

# Funktion zum Auffüllen von Nullen
def nullen_auffuellen_verbessert(alter_wert, neuer_wert):
    alter_wert_str = str(alter_wert)
    neuer_wert_str = str(neuer_wert)

    laengen_differenz = len(alter_wert_str) - len(neuer_wert_str)

    # Wenn der neue Wert weniger Ziffern hat, werden fehlende Ziffern
von links aufgefüllt
    if laengen_differenz > 0:
        neuer_wert_str = alter_wert_str[:laengen_differenz] +
neuer_wert_str
        neuer_wert = int(neuer_wert_str)

    return neuer_wert

# Funktion zur Auswahl des besten Segments bei zu vielen Ziffern
def bestes_segment_finden(alter_wert, neuer_wert_str, segment_laenge):
    # Generieren aller möglichen Segmente der gegebenen Länge
    moegliche_segmente = [neuer_wert_str[i:i+segment_laenge] for i in
range(len(neuer_wert_str) - segment_laenge + 1)]

    # Umwandeln der Segmente in Ganzzahlen
    moegliche_segmente = [int(segment) for segment in moegliche_seg-
mente]

    # Auswahl des Segments, das dem alten Wert am nächsten liegt
    bestes_segment = min(moegliche_segmente, key=lambda x: abs(x - al-
ter_wert))

    return bestes_segment

# Funktion zum Aktualisieren der Zählerdaten
def zaehlerdaten_aktualisieren(df, letzter_bekannter_wert, neuer_wert):
```

```
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    data = [[timestamp, neuer_wert]]
    df.loc[len(df)] = data[0]
    df.to_excel(ZAEHLER_DATEN_DATEI, index=False, header=True)
    return df

# Funktion zur Durchführung der OCR
def ocr_durchfuehren():
    # Zählerdaten aus der Excel-Datei laden
    df = pd.read_excel(ZAEHLER_DATEN_DATEI)

    # Letzten bekannten Zählerstand aus der Excel-Datei extrahieren
    letzter_bekannter_wert = int(df["Zählerstand"].iloc[-1])

    # Koordinaten für das Zuschneiden des Bildes festlegen
    links, oben, breite, hoehe = 65.0, 12.0, 155.0, 50.0

    # Bild herunterladen, drehen und zuschneiden
    jpg_daten, img = bild_herunterladen_drehen_und_zuschnei-
den(BILD_URL, ROTATIONS_GRAD, links, oben, breite, hoehe)

    # Ausgabe der Koordinaten des zugeschnittenen Bildes
    print(f"Zugeschnittene Bildkoordinaten: Links={links}, Oben={oben},
Breite={breite}, Hoehe={hoehe}")

    # Anzeigen des zugeschnittenen Bildes
    plt.imshow(img)
    plt.show()

    # Senden der API-Anfrage
    ergebnis = api_anfrage_senden(API_URL, API_KEY, SPRACHE, jpg_daten)

    # Verarbeitung des API-Ergebnisses
    for parsed_ergebnis in ergebnis.get("ParsedResults", []):
        parsed_text = parsed_ergebnis.get("ParsedText", "")
        neuer_wert_str = ''.join(filter(str.isdigit, parsed_text))  #
Behalten nur der Ziffern
        if neuer_wert_str:
            # Bei zu vielen Ziffern, finde das beste Segment
            if len(neuer_wert_str) > len(str(letzter_bekannter_wert)):
                neuer_wert = bestes_segment_finden(letzter_bekann-
ter_wert, neuer_wert_str, len(str(letzter_bekannter_wert)))
            else:
                neuer_wert = int(neuer_wert_str)

            print(f"Vorheriger Zählerstand: {neuer_wert} kw/h")

            # Korrektur von Werten mit weniger Ziffern
            if len(str(neuer_wert)) < len(str(letzter_bekannter_wert)):
```

```
                neuer_wert = nullen_auffuellen_verbessert(letzter_be-
kannter_wert, neuer_wert)

            # Plausibilitätsprüfung und eventuelle Korrektur
            if neuer_wert < letzter_bekannter_wert or neuer_wert -
letzter_bekannter_wert > MAX_DIFFERENZ:  # Erhöhter Spielraum für Kor-
rekturen
                print(f"Unplausibler Wert erkannt: {neuer_wert}")
                continue

            print(f"Vorheriger Zählerstand: {letzter_bekannter_wert}
kw/h")
            print(f"Neuer Zählerstand: {neuer_wert} kw/h")

            df = zaehlerdaten_aktualisieren(df, letzter_bekannter_wert,
neuer_wert)
            letzter_bekannter_wert = neuer_wert

    print("Ergebnisse für die Excel-Datei:")
    print(df)

    print("API-Antwort:")
    print(ergebnis)

# Durchführung der OCR
while True:
    ocr_durchfuehren()
    time.sleep(TIMER_INTERVAL)
```

11. Anhang C

```
#include <bmm8563.h>
#include <camera_index.h>
#include <camera_pins.h>
#include <led.h>
#include "battery.h"
#include "esp_camera.h"
#include <WiFi.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include "esp_sleep.h"
#include "esp32-hal-gpio.h"
#include "camera_config.h"

volatile bool isCameraOn = false;

const char *ssid     = "";
const char *password = "";

void startCameraServer();



void setup() {
    Serial.begin(115200);
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);  // disable   detector
    bat_init();
    bat_hold_output();
    Serial.setDebugOutput(true);
    Serial.println();
    pinMode(2, OUTPUT);
    digitalWrite(2, HIGH);


    // camera init
    esp_err_t err = esp_camera_init(&config);
    if (err != ESP_OK) {
        Serial.printf("Camera init failed with error 0x%x", err);
        return;
    }

    sensor_t *s = esp_camera_sensor_get();
    // initial sensors are flipped vertically and colors are a bit saturated
    s->set_vflip(s, 1);        // flip it back
    s->set_brightness(s, 1);   // up the blightness just a bit
    s->set_saturation(s, -2);  // lower the saturation
```

```
    // drop down frame size for higher initial frame rate
    s->set_framesize(s, FRAMESIZE_QVGA);

    Serial.printf("Connect to %s, %s\r\n", ssid, password);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connected");

    startCameraServer();

    Serial.print("Camera Ready! Use 'http://");
    Serial.print(WiFi.localIP());
    Serial.println("' to connect");
}

void loop() {
    // put your main code here, to run repeatedly:
    delay(100);
    digitalWrite(2, HIGH);
    delay(100);
    digitalWrite(2, LOW);
}
```

12. Anhang D

Messwerte

https://docs.google.com/spreadsheets/d/1-AmSYWgw_gEF-vBlg2OjH30E7_DDBtVfJ/edit#gid=43747878

https://github.com/Vaessenlu/Stromzaehler-Thesis

13. Literatur

[1] „Strompreisentwicklung 2023: So entwickelt sich der Strompreis." https://www.verivox.de/strom/strompreisentwicklung/ (Zugriff am: 16. Mai 2023).

[2] „ChatGPT: Get instant answers, find inspiration, learn something new." https://chat.openai.com/c/7031a196-a225-4e45-aac1-96cd1bae0a84 (Zugriff am: 3. August 2023).

[3] „Bundesnetzagentur - Homepage - Ferrariszähler (analoger Stromzähler)." https://www.bundesnetzagentur.de/SharedDocs/A_Z_Glossar/F/Ferrariszaehler.html (Zugriff am: 16. Mai 2023).

[4] W. Krukowski, *Grundzüge der Zählertechnik: Ein Lehr- und Nachschlagebuch*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1930.

[5] „ENIT: Stromzähler Lexikon Teil 1 - Wie und wo richtig messen? | ENIT." https://enit.io/wissen/knowledgebase/stromzaehlerlexikon (Zugriff am: 6. September 2023).

[6] Verbraucherzentrale.de. „Smart Meter: Was Sie über die neuen Stromzähler wissen müssen | Verbraucherzentrale.de." https://www.verbraucherzentrale.de/wissen/energie/preise-tarife-anbieterwechsel/smart-meter-was-sie-ueber-die-neuen-stromzaehler-wissen-muessen-13275 (Zugriff am: 16. Mai 2023).

[7] O. D. Doleski, *Realisierung Utility 4. 0 Band 2: Praxis der Digitalen Energiewirtschaft Vom Vertrieb Bis Zu Innovativen Energy Services*. Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2020.

[8] „SmartMeter - Netz + Service GmbH | Netzbetreiber in Nordhessen." https://netzplusservice.de/fuer-kunden/zaehler/smartmeter/ (Zugriff am: 6. September 2023).

[9] „Overview — Tesseract 3cd485a843e521cb2c2344110d332ea69405f375 documentation." https://tesseract-docs.readthedocs.io/en/latest/_source/core/overview/index.html (Zugriff am: 16. Mai 2023).

[10] E. Alpaydın, *Maschinelles Lernen,* 3. Aufl. (De Gruyter Studium). Berlin, Boston: De Gruyter Oldenbourg, 2022. [Online]. Verfügbar unter: https://www.degruyter.com/isbn/9783110740196

[11] „Bildrauschen entfernen, online und kostenlos mit KI - MyEdit Denoise." https://myedit.online/de/photo-editor/denoise (Zugriff am: 7. August 2023).

[12] Wikipedia. „Bildrauschen." https://de.wikipedia.org/w/index.php?title=Bildrauschen&oldid=228352806 (Zugriff am: 6. Januar 2023).

[13] „Codierung von Bilddaten - Weiterbildung Informatik." https://weiterbildung-informatik.wollw.de/chapter2/part1/sec2/ (Zugriff am: 7. August 2023).

[14] L. Kehrer, „Konzeption einer Anwendung zur Bestimmung von Dokumententypen und Extraktion von spezifischen Inhalten mithilfe von OCR-Bibliotheken sowie deren prototypische Implementierung.," 2021.

[15] J. Sauvola und M. Pietikäinen, „Adaptive document image binarization," *Pattern Recognition*, Jg. 33, Nr. 2, S. 225–236, 2000. doi: 10.1016/S0031-3203(99)00055-2. [Online]. Verfügbar unter: https://www.sciencedirect.com/science/article/pii/S0031320399000552

[16] T. A. Runkler, *Data analytics: Models and algorithms for intelligent data analysis* (Springer eBook Collection). Wiesbaden: Springer Vieweg, 2020.

[17] H. Handels, *Medizinische Bildverarbeitung: Bildanalyse, Mustererkennung und Visualisierung für die computergestützte ärztliche Diagnostik und Therapie,* 2. Aufl. (SpringerLink Bücher). Wiesbaden: Vieweg+Teubner, 2009.

[18] Amazon Web Services, Inc. „Funktionen von Amazon Textract | AWS." https://aws.amazon.com/de/textract/features/?pg=ln&sec=hs (Zugriff am: 25. Juli 2023).

[19] „Free OCR API." https://ocr.space/OCRAPI (Zugriff am: 31. Juli 2023).

[20] GitHub. „esp-who/docs/en/get-started/ESP-EYE_Getting_Started_Guide.md at master · espressif/esp-who." https://github.com/espressif/esp-who/blob/master/docs/en/get-started/ESP-EYE_Getting_Started_Guide.md (Zugriff am: 25. Juli 2023).

[21] „ESP-EYE AI Board I Espressif." https://www.espressif.com/en/products/devkits/esp-eye/overview (Zugriff am: 16. Mai 2023).

[22] m5stack-store. „M5Stack UnitV2 - The standalone AI Camera for Edge Computing (SSD202D) TinyML." https://shop.m5stack.com/products/unitv2-ai-camera-gc2145 (Zugriff am: 16. Mai 2023).

[23] m5stack-store. „ESP32 PSRAM Timer Camera X (OV3660)." https://shop.m5stack.com/products/esp32-psram-timer-camera-x-ov3660 (Zugriff am: 16. Mai 2023).

Abbildungsverzeichnis