

Fachbereich Mathematik/Informatik
Studiengang Master Informatik

Masterarbeit

Entwicklung eines Systems
zum Erfassen, Analysieren und Darstellen
von Energieverbräuchen in privaten Haushalten

Artem Petrov
Matrikelnr.: 929471

28. Juli 2016

1. Gutachter: Prof. Dr. Oliver Vornberger
Arbeitsgruppe Medieninformatik, Universität Osnabrück
2. Gutachter: Prof. Dr.-Ing. Elke Pulvermüller
Arbeitsgruppe Software Engineering, Universität Osnabrück

Erklärung

Ich versichere, diese Arbeit ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Osnabrück, den 28. Juli 2016

(Vorname Nachname)

Zusammenfassung

Um den Energieverbrauch und die daraus resultierenden Kosten von privaten Haushalten unter Kontrolle zu halten und um eventuell notwendige Energiesparmaßnahmen rechtzeitig einleiten zu können, wird in dieser Masterarbeit ein Softwaresystem entwickelt, das eine automatische Ablesung eines Zählers (für Strom/Gas/Wasser usw.) ermöglicht. Anhand von gespeicherten Zählerständen werden diverse Verbrauchsstatistiken in grafischer Form erstellt, die mit weiteren Daten (zum Beispiel Wetterdaten) kombiniert werden können. Als Hardware wird dabei ein Raspberry Pi mit einem Kameramodul eingesetzt.

Abstract

In order to control energy usage as well as resulting energy cost in private households and to initialize cost-cutting measures if required, a software system was created that allows automatic meter reading (electricity, gas, water), as a part of this master thesis. On the basis of stored meter readings are various consumption charts created, which can be combined with additional data (f.e. weather report). As a hardware solution is a Raspberry Pi with a camera modul deployed.

Danksagung

Ich danke allen Personen, die mich bei der Fertigstellung dieser Masterarbeit unterstützten:

- Herrn Prof. Dr. Oliver Vornberger für die Vergabe des Themas der Arbeit und für seine Tätigkeit als Erstgutachter.
- Frau Prof. Dr.-Ing. Elke Pulvermüller für die Übernahme des Zweitgutachtens.
- Meinen Eltern, die mich während meines Studiums unterstützten.

Osnabrück, im Juli 2016

Artem Petrov

Inhaltsverzeichnis

Abbildungsverzeichnis	xi
I Aufgabenstellung und Übersicht über existierende Lösungen	1
1 Einleitung	3
1.1 Motivation	3
1.2 Angestrebte Ergebnisse	5
1.3 Aufbau der Arbeit	6
2 Existierende Lösungen	9
2.1 Smart Meter	9
2.2 Kommerzielle Lösungen	11
2.3 Open-Source-Lösungen	15
2.4 Verwandte Projekte	18
3 Möglichkeiten für Erfassung des Zählerstandes	21
3.1 Digitale Zähler	21
3.2 Magnetische und optische Sensoren	22
3.3 Nichtinvasive Stromsensoren	24
3.4 Optische Zeichenerkennung	24
II Komponenten des Systems	27
4 Hardwarekomponenten	29
4.1 Raspberry Pi	29
4.2 RaspiCam und weitere Teile	30
4.3 Aufbau des Systems	32
III Implementierung	35
5 Implementierung der Softwarekomponente	37
5.1 Eingesetzte Software-Bibliotheken	37

5.2	Systemarchitektur	40
5.3	Backend	41
5.3.1	Programmablauf	41
5.3.2	Klassenübersicht	43
5.3.3	Bilderfassung und Vorbereitung	45
5.3.4	Ziffernerkennung	51
5.3.5	Datenhaltung	53
5.3.6	Autorun	56
5.4	Frontend	58
5.4.1	Architektur	58
5.4.2	Grafische Verbrauchsstatistiken	58
5.4.3	Zählereinstellungen	64
5.4.4	Trainieren	66
5.4.5	Systemeinstellungen	69
5.4.6	Server	69
6	Zusammenfassung und Ausblick	73
6.1	Zusammenfassung	73
6.2	Ausblick	74
IV	Anhänge	75
A	Installationsanleitung	77
B	Glossar	81
	Literaturverzeichnis	85

Abbildungsverzeichnis

1.1	Entwicklung der Energiepreise	4
2.1	Smart Meter	10
2.2	Smart-Metering-System von Voltcraft	12
2.3	Der e.manager von nD-enerserve GmbH	13
2.4	SmartPi von nD-enerserve GmbH	13
2.5	Smappee	14
2.6	OpenEnergyMonitor	15
2.7	Die emonPi	16
2.8	Das emonTx	17
2.9	Der Volkszaehler	18
3.1	S0-Schnittstelle	22
3.2	Reed-Schalter	22
3.3	Infrarot-Schreib-Lesekopf	22
3.4	Schaltplan einer Infrarot-Lichtschanke	23
3.5	Befestigung der IR-Lichtschanke an einem Stromzähler	23
3.6	Reflektierende Stelle am letzten Gaszählerzahlenrad	23
3.7	Befestigung der IR-Lichtschanke an einem Wasserzähler	23
3.8	Nichtinvasiver Stromsensor	24
3.9	Befestigung eines nichtinvasiven Stromsensors	24
3.10	Anschluss eines NI-Stromsensors an einen Einplatinenrechner	25
3.11	EnergyCam von Fast Forward AG	25
4.1	Raspberry Pi Version 2 Model B	30
4.2	Kameramodul für Raspberry Pi	31
4.3	Aufbau des Systems	32
4.4	Kameragehäuse	33
4.5	Leselampe	33
5.1	Systemarchitektur	40
5.2	Programmablauf	42
5.3	Klassendiagramm der Backend-Komponente	44
5.4	Methode capture der Camera-Klasse	45

5.5	Raspberry Pi GPIO	45
5.6	Ausgangsbild	46
5.7	MeterImage-Konstruktor	46
5.8	Methode getPreparedImage von MeterImage	47
5.9	Methode rotate von MeterImage	47
5.10	Mit Canny-Algorithmus gefundene Kanten	48
5.11	Methode getCannyEdges von MeterImage	48
5.12	Methode getLines von MeterImage	48
5.13	Methode getRotationAngle von MeterImage	49
5.14	Horizontale Linien	49
5.15	Alle Hüllkonturen	49
5.16	Methode findDigits von MeterImage	50
5.17	Gefilterte Hüllkonturen	50
5.18	Gefundene Ziffern	50
5.19	Methode trainAndStoreData von SingleDigitKNNRecogniser	51
5.20	Methode recognize von SingleDigitKNNRecognizer	52
5.21	Datenbankmodell	53
5.22	Methode updatePeriodicConsumption von MongoDataBaseManager	55
5.23	MapReduce-Verfahren in der Methode getWeather	56
5.24	Datei runner.py	56
5.25	Sequenzdiagramm	59
5.26	Grafische Verbrauchsstatistiken	60
5.27	Anpassung von Grafiken	61
5.28	Funktion loadChartDataAndRenderAllCharts	62
5.29	Funktion renderChart	63
5.30	Meter Settings	64
5.31	Meter Image Settings	64
5.32	OCR Setting	65
5.33	Zähleraufnahmen	66
5.34	Korrekt erkannter Zählerstand	67
5.35	Nicht erkannte Ziffer	67
5.36	Systemeinstellungen	69
5.37	Methode getWeather in server.py	70
5.38	Methode renderImages in server.py	70
5.39	HTML-Schablone images.html	71
5.40	Datei manage.py	72

Teil I

Aufgabenstellung und Übersicht über existierende Lösungen

Kapitel 1

Einleitung

*Der Computer wurde zur Lösung von Problemen erfunden,
die es früher nicht gab.*

BILL GATES

An dieser Stelle wird neben einer Motivation zum Thema Energieverbrauch und den damit verbundenen Ausgaben ein Überblick über die zu erreichenden Ziele sowie den Aufbau der vorliegenden Arbeit gegeben.

1.1 Motivation

Die Energieträger unterliegen seit geraumer Zeit starken Preisschwankungen (DeStatis, 2016a). Trotz der Tatsache, dass seit dem Jahr 2006 beim Erdgasbörsenpreis ein Rückgang um ca. 50 % beobachtet wird (Finanzen.net, 2016), lässt sich ein tendenzieller Energiepreisanstieg feststellen. Abbildung 1.1 auf der nächsten Seite verdeutlicht die Energiepreisentwicklung seit dem Jahr 2002.

Das Schlagwort Energiewende sollte inzwischen jedem bekannt vorkommen. Die Investitionen in Anlagen für erneuerbare Energien, in moderne Stromnetze, Speicher und Produktionstechniken sollten laut Bundesregierung bis zur Mitte des Jahrhunderts bis zu 550 Milliarden Euro betragen (Bundesregierung, 2016). Aus welchen Quellen werden diese enormen Ausgaben gedeckt?

„Machen die erneuerbaren Energien den Strom teurer? Nein, das Erneuerbaren-Energien-Gesetz senkt den Strompreis: Dadurch, dass der Strom aus regenerativen Energien garantiert abgenommen wird, sinkt der Börsenstrompreis insgesamt – denn dem größeren Stromangebot steht eine gleichbleibende Nachfrage gegenüber. [...] Allerdings: Der sinkende Börsenpreis hat auf die EEG-Umlage den gegenteiligen Effekt. [...] *Deshalb kann insgesamt der Strompreis für Verbraucher steigen.*“ (Bundesregierung, 2016)

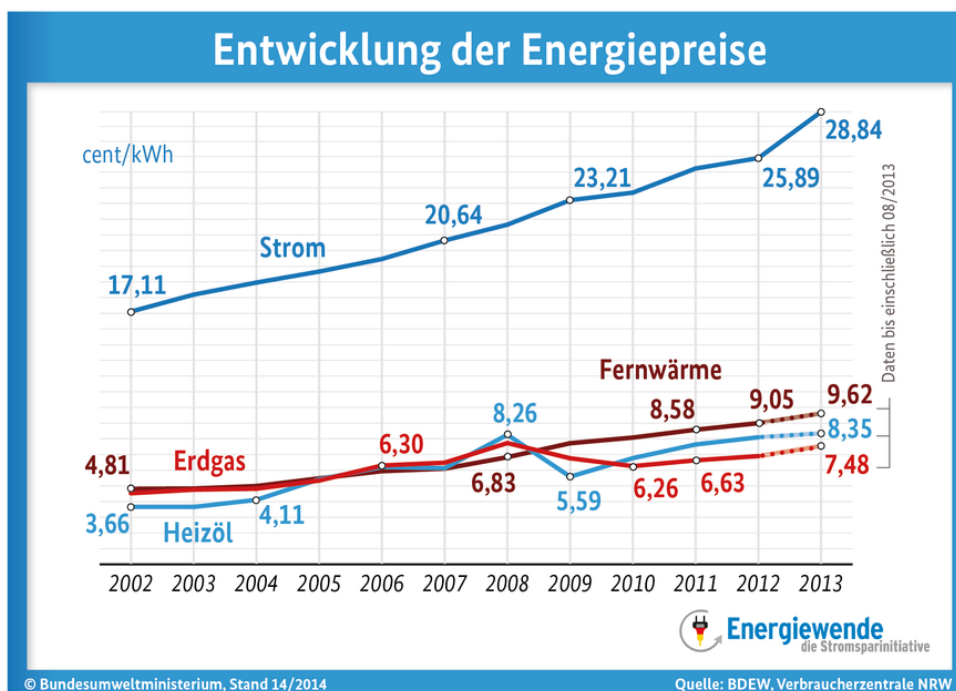


Abbildung 1.1: Entwicklung der Energiepreise

Die Börsenpreise für Energie haben kaum eine Auswirkung auf die Energieausgaben der privaten Haushalte. Auch wenn der Primärenergieverbrauch bis zum Jahr 2050 um 50 % gegenüber 2008 gesenkt werden soll (Stromsparinitiative, 2016), bleibt die Frage offen, ob private Haushalte in der Zukunft weniger Geld für Strom, Gas und Wasser ausgeben werden. Trotz steigender Energiepreise bleibt der Energieverbrauch nahezu konstant (DeStatis, 2008). Im Jahr 2008 betragen die Ausgaben privater Haushalte für die Wohnenergie (Strom, Gas, flüssige und feste Brennstoffe, Fernwärme) bis zu 5 % der Gesamtausgaben (DeStatis, 2008). Und in Euro ca. 1.600 € im Jahr.

Wie können die Energieausgaben gesenkt werden? Selbstverständlich durch die Preis senkung. Da das jedoch nicht der Fall ist, was die obigen Ausführungen verdeutlichen sollen, bleibt nur eine Möglichkeit. Der Energiekonsum muss verringert werden. Im Wesentlichen wirken auf den Konsum zwei Faktoren ein: zum einen die Energieeffizienz der Verbraucher und zum anderen das persönliche Konsumverhalten. Eine energetische Sanierung (zum Beispiel durch Anschaffung von verbrauchsarmen elektronischen Geräten oder sogar Neuinstallation einer modernen Brennwertheizung) stellt eine radikale Lösung dar, ist aber ohne großen wirtschaftlichen Aufwand kaum möglich. Aus wirtschaftlicher Sicht wird sich eine solche Investition erst nach mehreren Jahren auf die Ausgaben für Energie positiv auswirken. Das persönliche Konsumverhalten lässt sich jedoch ohne erheblichen finanziellen Ausgaben leicht beeinflussen. Aus der Lehre der Wirtschaftswissenschaften ist bekannt, dass eine kontinuierliche Kontrolle der Ausgaben, die auf der Aufzeichnung der Geldausgaben und der Analyse dieser Aufzeichnungen basiert, die Höhe der Geldausgaben

reduzieren lässt (Schmolke u. a., 2013). Dieses Prinzip kann auch auf das Konsumverhalten ausgeweitet werden. Das bestätigen mehrere Studien und Projekte.

Bereits 2009 ging eine Lösung namens **PowerMeter** des Internetgiganten Google online (Wikipedia, 2016a). Das Projekt wurde jedoch im Jahr 2011 aus mehreren Gründen (Fehrenbacher, 2011) beendet. Es wurde allerdings als wichtige Erkenntnis gewonnen, dass sich das Konsumverhalten durch eine ständige Kontrolle und Beobachtung des Energieverbrauchs verändern lässt. Andere Quellen bestätigen dies. Aus einem Projekt, an dem unter anderem *das Fraunhofer-Institut für Solare Energiesysteme ISE* und *das Fraunhofer-Institut für System- und Innovationsforschung ISI* beteiligt waren, geht hervor, dass durch Smart Metering bereitgestellte zeitnahe Informationen zum Energieverbrauch eine Ersparnis von 3,7 % (im manchen Fällen sogar bis zu 9,5 %) ermöglichen (Fraunhofer, 2011). Momentan werden viele Lösungen angeboten, die zur Analyse des Energieverbrauchs privater Haushalte herangezogen werden können. Doch viele davon haben sowohl Vor- als auch Nachteile. Über einige von diesen Ansätzen wird in **Kapitel 2** diskutiert.

Im Rahmen dieser Masterarbeit wird ein System entwickelt, das möglichst viele Vorteile bereits existierender Lösungen umschließt, kostengünstig, ausfallsicher und benutzerfreundlich ist und auch von Personen ohne tiefgehende Kenntnisse in den Bereichen Elektrotechnik und Informatik nachgebaut und im eigenen Haushalt in Betrieb genommen werden kann.

1.2 Angestrebte Ergebnisse

Das Ziel dieser Arbeit besteht darin, ein System zu entwickeln, das hauptsächlich für private Haushalte bestimmt ist. Mit dem System sollte es Privatverbrauchern möglich sein, die Energiezählerstände automatisch und kontinuierlich zu erfassen und die Daten dauerhaft zu speichern. Die gespeicherten Verbrauchsdaten sollen für analytische Zwecke verwendet werden. Anhand von aufgezeichneten Zählerständen sollen diverse Grafiken erstellt werden, die neben eigentlichen absoluten Zahlen zum Verbrauch auch einige zusätzliche Informationen enthalten. Bei einem Gaszähler sollen zum Beispiel neben dem periodischen Gasverbrauch ausgedrückt in Kubikmetern auch der entsprechende Verbrauch in Kilowattstunden und die daraus resultierenden Kosten angezeigt werden, die sich aus dem Absolutverbrauch ausrechnen lassen.

Da die Verbrauchsdaten über lange Perioden persistent gespeichert werden, sollen auch weitere Daten für Analysen herangezogen werden, die in unmittelbarem Zusammenhang mit der Konsumhöhe stehen. Bei dem Gasverbrauch über ein Jahr wäre der Außentemperaturverlauf des Jahres von besonderem Interesse. Durch die Berücksichtigung der Außentemperatur ergibt sich aus dem absoluten Energieverbrauch der sogenannte *temperaturbereinigte Energieverbrauch*, der auch in den Daten des Statistischen Bundesamts vorkommt (DeStatis, 2016b). Schließlich kann der Eigenverbrauch mit dem Bundes- oder Landesdurchschnittsverbrauch der Periode verglichen werden. Dieser Vergleich kann unter anderem bei Entscheidungen zur energetischen Sanierung behilflich sein.

Ein anderes Szenario stellt die Analyse von Auswirkungen des persönlichen Konsumverhaltens auf die Konsumhöhe der Energiegüter dar. Die Behauptung *Wenn die Temperatur*

nur um 1 Grad Celsius gesenkt wird, spart das rund 6 Prozent Energie kommt häufig in diversen Sparratgebern vor. Doch wie kann der Verbraucher feststellen, dass die Senkung der Zimmertemperatur in den kalten Wintertagen zu dem versprochenen Ersparnis führt, ohne dafür unnötig frieren zu müssen? Von vielen Energielieferanten werden Abrechnungen zum Energieverbrauch im jährlichen Turnus erstellt. Somit kommt die Rückmeldung etwas verzögert. Außerdem lässt sich nicht eindeutig beurteilen, ob die – wenn überhaupt – erzielte Ersparnis auf das Sparverhalten des Konsumenten zurückzuführen sind. Andere Faktoren, wie zum Beispiel das mildere Wetter, können viel größeren Einfluss auf den jährlichen Energieverbrauch haben. Das Problem wird beseitigt, indem die Verbrauchsdaten in sehr kurzen Zeitabständen erfasst werden und dem Privatverbraucher live angezeigt werden. Man könnte das Gedankenspiel fortsetzen und andere Anwendungsbeispiele der Datenanalyse vorstellen, die vom in dieser Arbeit entwickelten System bereitgestellt werden.

Nachfolgend werden die Anforderungen an das zu entwickelte System aufgeführt. Einerseits soll es sowohl in der Anschaffung als auch im Dauerbetrieb kostengünstig sein. Das Primärziel besteht in der Bereitstellung von Verbrauchsdaten, deren Analyse im Idealfall zu einer Energieersparnis und somit zur Reduzierung der Haushaltsausgaben für Energieträger führen soll. Dabei soll das System selbst keine erheblichen Kosten verursachen. Andererseits wird das System über einen langen Zeitraum eingesetzt und befindet sich im ständigen Betrieb. Daher muss es aus qualitativ hochwertigen Hardwarekomponenten bestehen und ausfallsicher sein.

Auch der Softwareteil des Systems muss diesen Anforderungen entsprechen und in der Lage sein, die Daten über mehrere Monate fehlerfrei aufzuzeichnen und persistent zu speichern. Außerdem soll es möglich sein, eine Sicherungskopie der gespeicherten Daten anzulegen und diese später wieder in das System einzuspielen. Die Softwarekomplexität darf für den Endnutzer des Systems nicht abschreckend wirken. Das System muss dagegen über eine benutzerfreundliche grafische Oberfläche verfügen, die auch für unerfahrene Personen intuitiv und selbsterklärend sein soll.

Des Weiteren soll das System aus frei verfügbaren Hardwarekomponenten aufgebaut sein, damit beim Nachbau keine Schwierigkeiten entstehen. Um den Systemnachbau weiterhin möglichst zu erleichtern, ist im **Anhang A** dieser Masterarbeit eine ausführliche Inbetriebnahmeanleitung enthalten. Der Quellcode des Softwareteils des Systems sollte unter einer Open-Source-Lizenz der Öffentlichkeit zur Verfügung gestellt werden, um eine Weiterentwicklung des Systems zu ermöglichen.

1.3 Aufbau der Arbeit

Der Aufbau dieser Arbeit gliedert sich in die folgenden Abschnitte: In **Kapitel 2** werden existierende Lösungen für das Erfassen von Zählerständen dargestellt. Dabei werden sowohl kostenpflichtige proprietäre Angebote als auch kostenlose Open-Source-Systeme betrachtet.

In **Kapitel 3** wird ein detaillierter Überblick über diverse Möglichkeiten zur Erfassung des Zählerstands gegeben. Dabei werden verschiedene Sensorarten dargestellt. In **Kapitel 4** werden die Komponenten des Hardwareteils vom in dieser Arbeit entwickelten System

beschrieben. Das Kapitel enthält eine Kostenübersicht sowie die Beschreibung des Systemaufbaus. In **Kapitel 5** wird die Implementierung der Softwarekomponenten des Systems dargelegt. Es werden die verwendeten Bibliotheken sowie alle Teile der Software aufgezählt. In **Kapitel 6** erfolgt eine Zusammenfassung sowie ein kurzer Ausblick hinsichtlich der Weiterentwicklung des Monitoring-Systems. Neben den genannten Kapiteln existieren in dieser Arbeit eine Reihe von Anhängen: In **Anhang A** sind ausführliche Informationen über die Inbetriebnahme des Monitoring-Systems zu finden, das im Rahmen dieser Arbeit entwickelt wurde.

Für das bessere Verständnis werden gebräuchliche englischsprachige Begriffe als *Terminus technicus* beibehalten und nicht ins Deutsche übersetzt. Ein Glossar, in dem wichtige Begriffe erläutert werden, befindet sich in **Anhang B**. Geschützte Warennamen und Warenzeichen sind innerhalb der Arbeit nicht gesondert kenntlich gemacht. Aus dem Fehlen eines solchen Hinweises kann demnach nicht geschlossen werden, dass es sich um einen freien Warennamen oder ein freies Warenzeichen handelt.

Kapitel 2

Existierende Lösungen

*Wir können das Rad nicht neu erfinden.
Wir können nur noch ein wenig daran drehen.*

MARGOT S. BAUMANN

Im folgenden Kapitel werden existierende Lösungen vorgestellt, die zur Erfassung und Analyse vom Energieverbrauch in privaten Haushalten eingesetzt werden können. Neben dem bereits erwähnten Smart Metering und manchen Projekten, die darauf basieren, werden sowohl einige Open-Source- als auch proprietäre Systeme beschrieben. Außerdem werden die Vor- und Nachteile dieser Systeme betrachtet.

2.1 Smart Meter

Als Erstes wird in diesem Abschnitt der Begriff *Smart Meter* definiert.

Definition 2.1.1 (Smart Meter)

„Ein Smart Meter, auch im deutschsprachigen Raum intelligenter Zähler genannt, ist ein Zähler für Energie, z. B. Strom oder Gas, der dem jeweiligen Anschlussnutzer den tatsächlichen Energieverbrauch und die tatsächliche Nutzungszeit anzeigt und in ein Kommunikationsnetz eingebunden ist.“ (Wikipedia, 2016b)

Solche Smart Meter (Abbildung 2.1, S. 10) sind bereits seit den 1990er-Jahren vor allem für Großkunden verfügbar. Seit ungefähr 2010 werden intelligente Zähler auch in Privathaushalten installiert. Die in regelmäßigen Abständen erfassten Zählerstände (meistens alle 15 Minuten) werden im Zähler zwischengespeichert und automatisch an das Energieversorgungsunternehmen übertragen.

Smart Meter sind in das sogenannte *Smart Grid* (intelligente Stromnetz) integriert. Um die Energie möglichst effizient nutzen zu können, geht der Trend der Energieerzeugung in



Abbildung 2.1: Smart Meter

Richtung dezentraler Erzeugungsanlagen. Alle Akteure auf dem Energiemarkt (vor allem Verbraucher und Erzeuger) werden in einem intelligenten Stromnetz zusammengefasst. Die Lastregelung in einem Smart Grid erfolgt dynamisch. Dadurch können unter anderem wetterabhängige Energieerzeuger (zum Beispiel Windkraftanlagen) problemlos in das Netz integriert werden. Die von ihnen erzeugte Energie wird fast in vollem Umfang – dank dynamischer Regelung – verbraucht. Die Smart Meter sind in erster Linie für die Regelung solches eines intelligenten Netzes bestimmt. Als Nebeneffekt können die gesammelten Verbrauchsdaten dem Endkunden über ein Webportal in visueller Form dargestellt werden. Obwohl das Erfassen und das Übertragen von Zählerständen sehr genau erfolgt und ziemlich zuverlässig ist, hat der Einsatz eines Smart Meters für Zwecke der Verbrauchsanalyse in privaten Haushalten einige Nachteile. Zum einen sind das hohe Anschaffungskosten.

Zum Beispiel bietet die Stadtwerke Osnabrück AG ihren Kunden zwei Produkte namens **smartWeb** und **smartCOCKPIT** an (SWO, 2016). Bei **smartWeb** handelt es sich um ein Webportal, in dem dem Nutzer eine grafische Übersicht des Stromverbrauchs dargestellt wird. Das andere Produkt – **smartCOCKPIT** – ist eine Smartphone-App mit gleicher Funktionalität. Um das Angebot nutzen zu können, muss ein Smart Meter installiert werden, was einmalige Kosten in Höhe von 66,28 € verursacht. Ein monatliches Abo für **smartWeb** kostet 7,44 € und für **smartCOCKPIT** 8,20 €. Jährlich fallen somit 89,28 € beziehungsweise 98,40 € an. Es lässt sich leicht ausrechnen, dass bei einem Stromverbrauchspreis von 23,65 Cent/kWh durch den Einsatz von **smartWeb** 377 kWh eingespart werden müssen, damit sich das Abo lohnt. In dieser Rechnung werden Einbaukosten für einen neuen Stromzähler nicht berücksichtigt.

Laut Studien des Fraunhofer-Instituts für System- und Innovationsforschung, die im **Kapitel 1.1** erwähnt wurden, lässt sich durch die Bereitstellung zeitnaher Informationen zum Energieverbrauch eine Ersparnis in Höhe von ca. 4 % erzielen. Somit muss der jähr-

liche Verbrauch über 9500 kWh liegen. Das ist doppelt so hoch wie der durchschnittliche Verbrauch einer vierköpfigen Familie. Kurz zusammengefasst: Der Einsatz von smartWeb, smartCOCKPIT und ähnlichen Produkten anderer Energielieferanten ist für einen durchschnittlichen Haushalt unwirtschaftlich.

Des Weiteren werden momentan Smart Meter fast ausschließlich für die Messung des Stromverbrauchs installiert. Der Einbau von Smart Meter zur Erfassung des Gasverbrauchs ist zwar möglich, wird aber von keinem Energielieferanten standardmäßig angeboten. Laut Informationen von einem Mitarbeiter der Stadtwerke Osnabrück ist der massenhafte Einbau von intelligenten Gaszählern in absehbarer Zukunft nicht geplant.

Schließlich werden die Verbrauchsdaten in verschlüsselter Form an den Energielieferanten übertragen. Der Endnutzer hat keine Möglichkeit, auf diese Daten zuzugreifen, abgesehen von der Benutzung eines kostenpflichtigen Angebots. In manchen Ländern (zum Beispiel in Spanien) werden die alten Ferraris-Zähler nach und nach vom Energieversorger durch Smart Meter ersetzt. Laut Planung sollen alle alten Zähler in Spanien bis zum Jahr 2018 ausgetauscht werden (Iberdrola, 2016). Für den Endverbraucher fallen dabei keine zusätzlichen Kosten an. Auch in Deutschland ist der Einbau von Smart Meter sowohl für Strom als auch für Gas in manchen Fällen gesetzlich vorgeschrieben (Juris, 2005).

2.2 Kommerzielle Lösungen

Angesichts steigender Energiepreise stößt das Thema Energieverbrauchs-messung auf positive Resonanz. Das erkannten viele große und kleine Unternehmen, die in IT- und Elektrotechnikbereichen agieren. Momentan werden viele sogenannte *Energiemanagement-* und *Energiemonitoring-Systeme* angeboten, die sowohl alleinstehend als auch als ein Bestandteil eines *Smart-Home-Systems* konzipiert sind. Alle Systeme zur Erfassung des Verbrauchs können grob nach Art der Erfassung in zwei Kategorien unterteilt werden: Systeme mit *zentraler* und *dezentraler* Erfassung.

Systeme, wie zum Beispiel **e.manager** von nD-enerserve GmbH (nDenerserve, 2016) oder das **Smart-Metering-System** von Voltcraft (Abbildung 2.2, S. 12)¹, messen den Gesamtstromverbrauch eines Haushalts. Erfasst wird dabei der Zählerstand oder der momentane Verbrauch, gemessen an der Stromeingangsleitung. Das Smart-Metering-System hat dabei einen erheblichen Nachteil, da für den Einbau und Inbetriebnahme des Systems ein Eingriff in die bestehende Installation durchgeführt werden muss. Dabei wird im Verteilerschrank nach dem herkömmlichen Stromzähler ein elektronischer Stromzähler eingebaut. Über eine Funkschnittstelle liefert dieser alle Verbrauchsdaten an einen USB-Funkstick, der an einen PC angeschlossen werden kann.

Des Weiteren kann die Auswertung der Verbrauchsstatistiken nur auf einem einzelnen Rechner geschehen. Um über das Internet auf die Daten zugreifen zu können, bedarf es einer weiteren Lösung, die nicht im Lieferumfang enthalten ist. Das System kostet 236,95 €. Dazu fallen noch zusätzliche Kosten für den Einbau an. Auch die Kunden sind laut Produktbewertungen vor allem mit der Softwarekomponente des System nicht zufrieden.

¹<https://www.conrad.de/de/beratung/energie/smart-metering/smart-metering.html>



Abbildung 2.2: Smart-Metering-System von Voltcraft

Eine andere Lösung namens e.manager (Abbildung 2.3, S. 13) verfolgt einen anderen Ansatz. Der Stromverbrauch wird in diesem Fall durch sogenannte *nichtinvasive Stromsensoren* gemessen. Wie eine solche Messung funktioniert, wird in **Kapitel 3.3** beschrieben. Dabei handelt es sich bei e.manager um ein System, das nicht nur für Zwecke der Verbrauchserfassung eingesetzt wird. Es ist ein *Hausautomationssystem* mit umfangreicher Funktionalität. Dementsprechend kann das System für einen ziemlich hohen Preis erworben werden. Eine Basiskonfiguration ohne Stromerfassung und ohne WLAN-Einbindung kostet 605 €.

Das Unternehmen bietet ein weiteres Produkt unter dem Namen **SmartPi** an ². Dabei handelt es sich um ein Erweiterungsmodul für den Einplatinenrechner **Raspberry Pi** (s. **Kapitel 4.1**). Mit dem SmartPi (Abbildung 2.4, S. 13) können sowohl Strom als auch Spannung gemessen werden. Eine Open-Source-Software ermöglicht einen weltweiten Zugriff auf die gesammelten Verbrauchsdaten. Allerdings ist momentan keine grafische Benutzeroberfläche vorhanden, stattdessen werden die Daten über eine REST-Schnittstelle ausgeliefert ³. Außerdem ist die Software fehlerbehaftet und nicht ausgereift. Die Entwicklung dieses Produkts wurde unter anderem durch ein Projekt auf der *Kickstarter-Plattform* finanziert (Kickstarter, 2016). SmartPi kann für 124,95 € ab dem 31. Juli 2016 erworben werden. Das Modul wird unter einer Open-Source-Lizenz entwickelt. Somit ist der Einsatz von SmartPi zur Erfassung des Stromverbrauchs durchaus denkbar.

Bei manchen Produkten für Energiemonitoring, die hier nicht erwähnt wurden, erfolgt die Erfassung des Verbrauchs dezentral. Dabei werden die einzelnen Verbraucher mittels

²<https://shop.enerserve.eu/detail/index/sArticle/262/sCategory/5>

³<https://github.com/nDenerserve/SmartPi/>

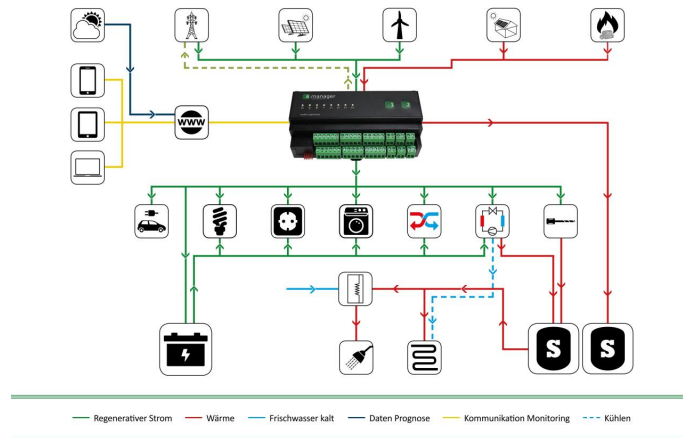


Abbildung 2.3: Der e.manager von nD-enerserve GmbH

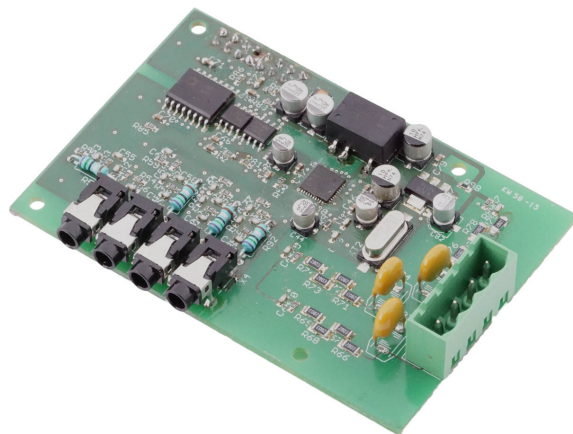


Abbildung 2.4: SmartPi von nD-enerserve GmbH

einer Zwischensteckdose, die über ein Mess- und Funkmodul verfügt, an das Stromnetz angeschlossen. Die gemessenen Verbräuche werden in einem zentralen Modul gesammelt und über eine Weboberfläche visualisiert. Ein Nachteil bei diesem Vorgehen besteht darin, dass der Gesamtverbrauch nicht erfasst werden kann und manche Großverbraucher (zum Beispiel ein Elektroherd) nicht über eine zweipolige Steckdose versorgt werden. Diese Lösung hilft jedoch beim Auffinden der Geräte mit einem übermäßigen Stromverbrauch.

Bei allen in diesem Unterkapitel zuvor betrachteten Produkten handelt es sich um Energiemesssysteme, die ausschließlich für die Messung des Stromverbrauchs bestimmt sind. Eine bedingt ausgereifte kommerzielle Lösung für Privatanwender zum Erfassen von Strom-, Gas- und Wasserzählern stellt ein Produkt namens **Smappee** dar (Smappee, 2016).

Von Smappee (Abbildung 2.5) werden jedoch nicht die eigentlichen Zählerstände erfasst, sondern der momentane Verbrauch. Dadurch ist die Erfassung des Verbrauchs bei einem Systemausfall nicht möglich. Zu den Vorteilen von Smappee zählen unter anderem vergleichsweise niedrige Anschaffungskosten (199 € für Energiemonitor und 119 € für Gas- und Wassermonitor), intuitive Benutzeroberfläche und modernes Hardwaredesign. Das Werbeversprechen des belgischen Herstellers hält die Smappee-Software jedoch nicht. Unter anderem soll Smappee automatisch einzelne Verbraucher anhand des Gesamtverbrauchs erkennen. Jedoch besteht laut zahlreichen negativen Kundenrezensionen⁴ an einigen Stellen der Software großes Verbesserungspotenzial. Des Weiteren werden alle Verbrauchsdaten auf den Servern des Systemherstellers zentral gespeichert. Aus Datenschutzsicht ist diese Lösung nicht zu empfehlen.



Abbildung 2.5: Smappee

⁴<https://play.google.com/store/apps/details?id=be.smappee.mobile.android>

Ein wesentlicher Kritikpunkt der vorgestellten Systeme sind deren ziemlich hohe Kosten. Des Weiteren muss der Privatanwender über tiefe Kenntnisse in unterschiedlichen Bereichen verfügen, um die Systeme – von Smappee abgesehen – einzubauen und zu betreiben. Kommerzielle Systeme können nur durch Komponenten desselben Herstellers erweitert werden. Diese Systeme sind stark zweckgebunden und der Datenaustausch erfolgt meist über proprietäre verschlüsselte Protokolle. Falls ein System vom Markt genommen oder der Hersteller insolvent wird, erhält der Kunde keine weitere Unterstützung. Das in dieser Masterarbeit zu entwickelnde Energiemesssystem soll möglichst frei von diesen Nachteilen sein.

2.3 Open-Source-Lösungen

Das Thema der Erfassung der Energie- und Wasserzähler beschäftigt weltweit viele Enthusiasten, die nicht mit den bestehenden kommerziellen Lösungen zufrieden sind und sich mit der Entwicklung eigener Monitoring-Systeme beschäftigen. Nachfolgend werden zwei bekannte Open-Source-Projekte vorgestellt.

OpenEnergyMonitor ist ein Open-Source-Projekt, das in England gegründet wurde. Eine ständige Stromverbrauchsüberwachung soll den Menschen dabei helfen, die elektrische Energie möglichst effizient nutzen zu können. Das Hauptziel ist ein möglichst großer Beitrag zur Nutzung der erneuerbaren oder emissionsfreien Energiequellen, um unter anderem den Ausstieg aus der Erzeugung von Atomenergie zu erleichtern (OpenEnergyMonitor, 2016a). OpenEnergyMonitor ermöglicht sowohl eine Echtzeitüberwachung des Stromverbrauchs als auch eine Analyse der Verbrauchsstatistiken über einen langen Zeitraum.

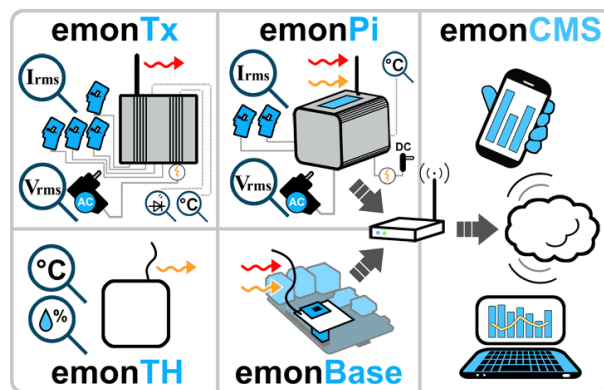


Abbildung 2.6: OpenEnergyMonitor

Das System (Abbildung 2.6) besteht aus einer Hardware- und einer Softwarekomponente. Beide Teile des Systems stehen unter einer freien Lizenz. Mittels nichtinvasiver Stromsensoren oder *Impulssensoren* (s. **Kapitel 3.2** und **Kapitel 3.3**) wird der momentane Stromverbrauch erfasst. Die Verbrauchsdaten werden auf einem *Webserver* in einer *MySQL-Datenbank* abgelegt. Schließlich werden die Daten in einem speziell dafür entwickelten *Content-Management-System* – *emonCMS* – in Form von Grafiken visualisiert. Die grafische Oberfläche kann weltweit abgerufen werden.

Die zentrale Komponente des Systems ist die *Basistation*. Aktuell wird diese auf dem Einplatinenrechner *Raspberry Pi* (s. **Kapitel 4.1**) aufgebaut und unter dem Namen **emonPi** vertrieben. Die Hauptkomponente verfügt über diverse Ein- und Ausgänge. Die Stromsensoren werden mit der Basistation über ein Kabel verbunden. Raspberry Pi besitzt ausreichend Leistung, sodass neben der Auslesung der Sensoren auch ein reibungsloser Betrieb eines Webservers (in diesem Fall kommt ein Open-Source-Webserver *Apache2* zum Einsatz) und eines *Datenbankservers* möglich ist. Der Minirechner ist wahlweise über eine WLAN- oder LAN-Schnittstelle mit dem Netzwerk verbunden. Die Seiten des Content-Management-Systems, das in der Programmiersprache *PHP* implementiert ist, können in einem *Webbrowser* sowohl an einem Laptop oder einem PC als auch auf einem mobilen Endgerät dargestellt werden. Auch andere Sensoren (zum Beispiel Temperatur-, Luftfeuchtigkeits- oder CO-Sensoren) können an die Basistation angeschlossen werden.



Abbildung 2.7: Die emonPi

Weltweit hat das Projekt viele Anhänger und die Support-Community wächst ständig (OpenEnergyMonitor, 2016b). Um OpenEnergyMonitor einsetzen zu können, muss der Benutzer über grundlegende Kenntnisse in den Bereichen Web- und PHP-Programmierung, Webserver-Administration, Datenbankmanagementsysteme und Elektrotechnik verfügen. Momentan wird eine fertige Lösung zum Kauf angeboten (Abbildung 2.7). Die Hardware ist dabei komplett zusammengebaut und die Software ist bereits vorinstalliert (OpenEnergyMonitor, 2016c).

Aber auch in diesem Fall muss der Benutzer dazu bereit sein, sich in die Grundlagen der oben erwähnten IT-Gebiete einzuarbeiten. Der Preis für das Komplettsystem mit einem Stromsensor (was die Verwendung auf Einphasenstromnetze begrenzt) beträgt momentan umgerechnet 230 €. Für die Dreiphasennetze wird ein RF-Modul (Abbildung 2.8, S. 17) zum Preis von umgerechnet 76 € angeboten. Der Aufbau des Moduls ist mit dem Aufbau des im vorherigen Abschnitt erwähnten SmartPi fast identisch. Die erfassten Daten werden in diesem Fall über einen 433-MHz-Funkkanal an die Basistation übertragen.



Abbildung 2.8: Das emonTx

Grundlegend ist OpenEnergyMonitor eine komplette, professionelle und ausgereifte Lösung, die in privaten Haushalten eingesetzt werden kann. Die Erfassung von Gas- oder Wasserzählerständen kann dabei mit einem Impulssensor erfolgen. Falls bei dem Benutzer Elektrotechnikenkenntnisse vorhanden sind, können alle Hardwarekomponenten selbstständig nachgebaut werden. Alle Schaltpläne und Leitplatten-Layouts sind frei verfügbar. Die Elektronikkomponenten des Systems (Mikrocontroller, Sensoren, aktive und passive Bauteile) können in vielen Elektronikfachgeschäften bezogen werden.

Eine – wenn auch keine gleichwertige – Alternative zu OpenEnergyMonitor stellt ein Projekt aus Deutschland dar, das unter dem Namen **Volkszaehler** bekannt ist (Volkszaehler, 2016b). Das Projekt befindet sich immer noch im Aufbaustadium.

„Volkszaehler ist keine Fertiglösung “von der Stange” (und kann und will das auch nicht sein)! Etwas Bereitschaft zum Basteln ist Voraussetzung! Kenntnisse im Umgang mit Linux, und ggf. auch Programmierung und Elektronik sind (gerade im aktuellen Stadium des Projekts) vorteilhaft.“ (Volkszaehler, 2016a)

Diese Aussage korreliert sehr schwach mit dem Motto des Projekts „*Volkszaehler.org – das Smart Meter für jeden*“.

Der Aufbau des Systems (Abbildung 2.9, S. 18) ähnelt dem Aufbau von OpenEnergyMonitor. Die von Sensoren erfassten Daten werden auf einen Webserver übertragen und können über ein Webinterface visualisiert werden. Als Webserver wird der Einplatinenrechner Raspberry Pi empfohlen. Alle Schaltpläne für die Hardwarekomponenten des Systems sind frei verfügbar. Somit kann das System von Benutzern mit Elektrotechnikenkenntnissen nachgebaut werden. Im Vergleich zu OpenEnergyMonitor hat Volkszaehler keine Vorteile. Es ist keine Möglichkeit gegeben, ein fertig zusammengebautes System zu erwerben. Die Softwarekomponente des Systems ist nicht ausgereift und fehlerbehaftet, was die Beiträge der Mailingliste des Projekts bestätigen ⁵. Die Webseiten des Projekts sind unübersichtlich organisiert. Die Software- und Hardwareokumentation ist unvollständig und häufig widersprüchlich.

⁵<http://volkszaehler.org/pipermail/>

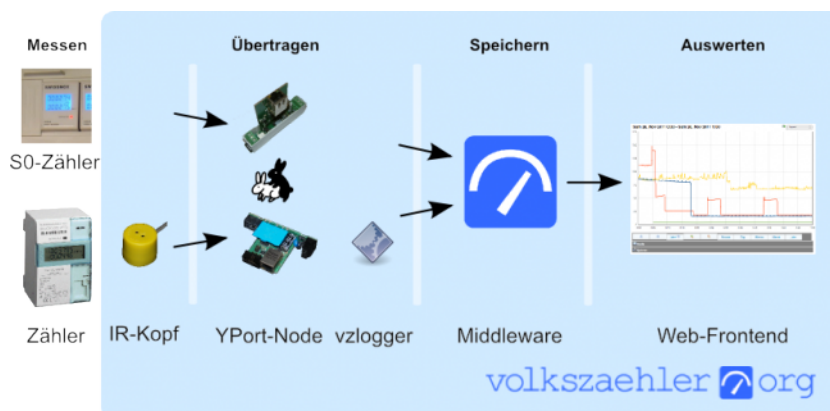


Abbildung 2.9: Der Volkszähler

2.4 Verwandte Projekte

Im Projekt *OpenCV Praxis: OCR für den Stromzähler* (Kompf, 2016) wurde ein Monitoring-System implementiert, um den Stromverbrauch zu erfassen. Als Hardware wurde der Einplatinenrechner Raspberry Pi mit einer Webkamera benutzt. Auf der Webseite des Projekts ist lediglich die Implementierung der Softwarekomponente dargestellt. Der Aufbau der Hardwarekomponente ist knapp beschrieben. Zur Beleuchtung wurde eine LED verwendet, die über ein zusätzliches 12-Volt-Netzteil mit Strom versorgt wird. Die Software ist in der Programmiersprache C++ implementiert und kann aus einem Git Repository heruntergeladen werden. Die Bildverarbeitung erfolgt mithilfe der Open-Source-Bibliothek *OpenCV* (s. **Kapitel 5.1**). Die Software wird über die Konsole gestartet und verwaltet. Das Ändern der Konfiguration der Software erfolgt durch das Editieren einer Konfigurationsdatei. Dafür ist keine grafische Benutzeroberfläche vorhanden. Die Grafiken werden ebenso über die Konsole erzeugt und angezeigt. Es gibt jedoch Hinweise, wie eine einfache Weboberfläche erstellt werden kann. Der in diesem Projekt verwendete Algorithmus der Ziffernidentifikation wurde für die Implementierung der Softwarekomponente dieser Masterarbeit benutzt.

Ein ähnliches Projekt, das auf der russischen Seite dargestellt ist (Schutkin, 2016), beschäftigt sich mit der Erfassung eines Gaszählers. Als Hardware kommt ein Android-Smartphone zum Einsatz. Die Bildverarbeitung wird ebenso mit OpenCV ausgeführt. Die Ziffernerkennung erfolgt mit einem lernfähigen Algorithmus. Die Software ist in der Programmiersprache *Python* (s. **Kapitel 5.1**) implementiert und schlecht strukturiert, da sich fast alle Methoden in einer gemeinsamen Datei befinden. Für das Speichern von Daten wird der *Cloud-Service Google Drive* benutzt. Es ist eine Weboberfläche vorhanden, in der die Verbrauchsstatistiken in grafischer Form dargestellt werden. Über die Oberfläche kann der OCR-Algorithmus trainiert werden. Die aus diesem Projekt stammenden Daten wurden für das Testen des Ziffernerkennungsalgorithmus verwendet, der in dieser Masterarbeit implementiert wurde.

Im nächsten Kapitel werden unterschiedliche Möglichkeiten zur Erfassung des Energieverbrauchs dargestellt. Neben den bereits erwähnten nichtinvasiven Stromsensoren und Impulssensoren werden auch andere Sensorarten und Ableseverfahren beschrieben.

Kapitel 3

Möglichkeiten für Erfassung des Zählerstandes

*Computer sind unbrauchbar.
Sie können nur Fragen beantworten.*

PABLO PICASSO

In diesem Kapitel werden diverse Möglichkeiten zur Erfassung eines Energie- oder Wasserzählers vorgestellt. Dabei wird auf unterschiedliche Sensorarten und ihre Funktionsweise eingegangen. Aus den betrachteten Alternativen zur Ablesung des Zählerstands wird schließlich eine ausgewählt, die in dieser Arbeit verwendet wird.

3.1 Digitale Zähler

Die einfachste und komfortabelste Lösung zum Erfassen des Energieverbrauchs stellt die sogenannte *S0-Schnittstelle* dar.

Definition 3.1.1 (S0-Schnittstelle)

„Die S0-Schnittstelle (gesprochen S-Null-Schnittstelle) ist eine Hardwareschnittstelle für die Übertragung von Verbrauchs-Messwerten in der Gebäudeautomatisierung. Die Übertragung der Daten erfolgt mit Hilfe von gewichteten Impulsen, d.h., pro Kilowattstunde oder Kubikmeter wird eine bestimmte Anzahl Impulse übertragen.“(Wikipedia, 2016c)

Der Verbrauch lässt sich somit trivial ermitteln. Die S0-Schnittstelle besitzt zwei Ausgänge: S0+ und S0- (Abbildung 3.1, S. 22). Beide Ausgänge werden mit einem Erfassungsgerät (zum Beispiel mit einem Raspberry Pi) verkabelt. Sobald ein Impuls ausgelöst wird, wird eine interne Programmvariable inkrementiert. Der Wert dieser Variable, multipliziert mit einem Faktor, der die verbrauchte Energiemenge pro Impuls angibt, stellt

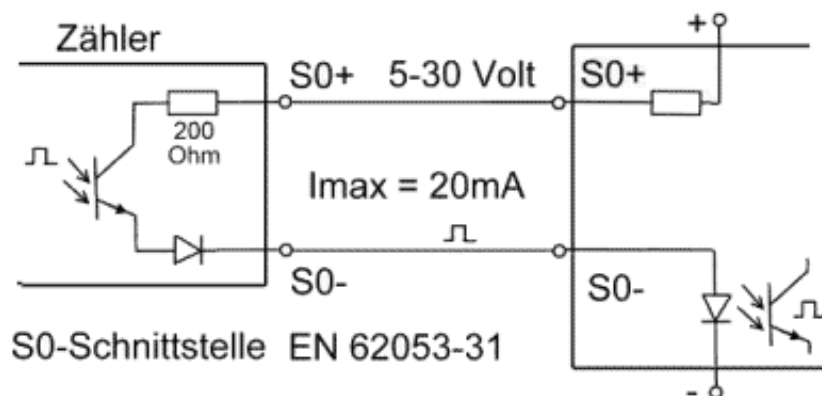


Abbildung 3.1: S0-Schnittstelle

den aktuellen Zählerstand dar. Der Zählerstand kann dann in periodischen Abständen in einer Datenbank abgelegt werden. Anschließend können die Verbrauchsdaten visualisiert und dem Benutzer angezeigt werden.

Viele Smart Meter verfügen über eine S0-Schnittstelle. In den meisten Fällen kann der Nutzer auf die S0-Schnittstelle des vom Energielieferanten eingebauten digitalen Zählers nicht zugreifen. Es muss also ein digitaler Zwischenzähler eingebaut werden. Allerdings führt das zu zusätzlichen Kosten und kann nicht immer problemlos erfolgen (vor allem in einer Mietwohnung).

3.2 Magnetische und optische Sensoren



Abbildung 3.2: Reed-Schalter



Abbildung 3.3: Infrarot-Schreib-Lesekopf

Von manchen Herstellern werden Strom-, Gas- und Wasserzähler angeboten, die eine andere Ablesemöglichkeit anbieten. In diesem Fall werden durch ein mechanisches Bauteil gewichtete magnetische Impulse ausgelöst. Das Ablesen erfolgt durch einen *Reed-Schalter* (Abbildung 3.2), dessen Kontakte durch ein Magnetfeld betätigt werden. Einige Zähler

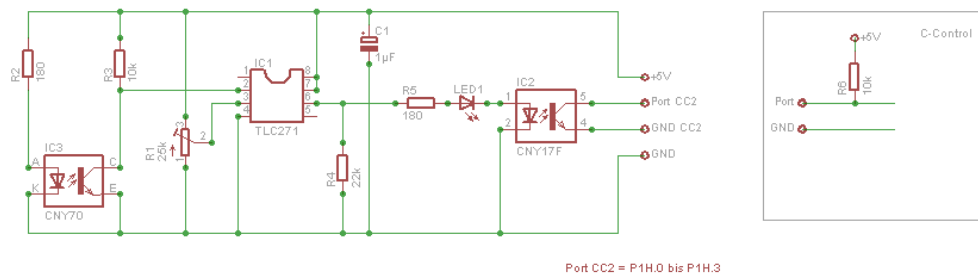


Abbildung 3.4: Schaltplan einer Infrarot-Lichtschranke

geben anstatt eines magnetischen einen gewichteten optischen Impuls aus. Der optische Impuls kann mit einem optischen Sensor, dem sogenannten *Infrarot-Schreib-Lesekopf* (Abbildung 3.3, S. 22), erfasst werden ¹.

Falls die Zählerbauart keinen Impulsgeber vorsieht, kann eine selbstgebaute *Infrarot-Lichtschranke* (Abbildung 3.4) verwendet werden. Die Lichtschranke kann aus wenigen Bauteilen auf einer Lochrasterplatte aufgebaut werden ². Bei einem Stromzähler muss die Lichtschranke auf die Drehscheibe ausgerichtet werden (Abbildung 3.5). Pro Scheibenumdrehung gibt die IR-Schranke einen Impuls aus. Um auf diese Art den Gasverbrauch zu erfassen, muss das Zählwerk des Gaszählers eine reflektierende Stelle haben (Abbildung 3.6). Meistens befindet sich ein reflektierender Aufkleber auf dem letzten Zahlenrad des Gaszählers. Auch der Wasserverbrauch lässt sich mit einer IR-Lichtschranke erfassen ³. Abbildung 3.7 soll das verdeutlichen.

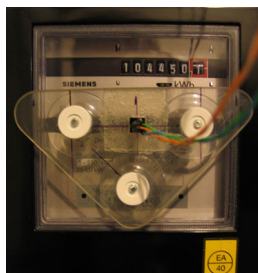


Abbildung 3.5: Befestigung der IR-Lichtschranke an einem Stromzähler

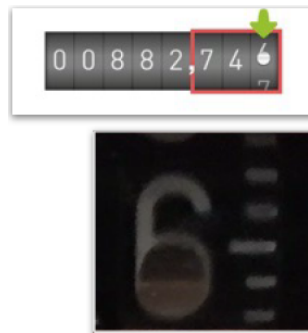


Abbildung 3.6: Reflektierende Stelle am letzten Gaszählerzahlenrad



Abbildung 3.7: Befestigung der IR-Lichtschranke an einem Wasserzähler

Die weitere Datenerfassung und Visualisierung unterscheidet sich nicht von der Verwendung eines Smart Meters mit S0-Schnittstelle.

¹<http://wiki.volkszaehler.org/hardware/controllers/ir-schreib-lesekopf>

²<http://www.hobbyheizer.de/index.php/haussteuerung/stromzaehler>

³<http://homematic-forum.de/forum/viewtopic.php?t=23744>

3.3 Nichtinvasive Stromsensoren

Der Verbrauch von elektrischer Energie kann mit den bereits erwähnten nichtinvasiven Stromsensoren (Abbildung 3.8) gemessen werden. Der Sensor wird an einer *Phasenleitung* im Sicherungskasten befestigt (Abbildung 3.9).



Abbildung 3.8: Nichtinvasiver Stromsensor

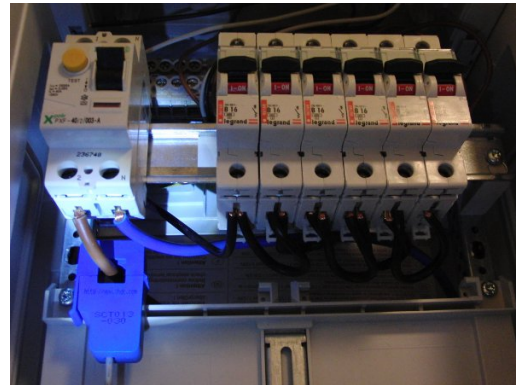


Abbildung 3.9: Befestigung eines nichtinvasiven Stromsensors

Der Sensor besteht aus einer Spule, in der ein *Induktionsstrom* erzeugt wird, dessen Höhe im fast linearen Verhältnis zur Höhe des durch die Leitung fließenden Stroms steht. Der Induktionsstrom wird mittels einer einfachen Schaltung (Abbildung 3.10, S. 25) in eine *Referenzspannung* umgewandelt, die schließlich durch einen *Analog-Digital-Wandler (ADC)* in einen numerischen Wert konvertiert wird. Aus diesem Wert lässt sich dann der momentane Stromverbrauch ausrechnen. Der Einplatinenrechner Raspberry Pi oder ein **Arduino-Board** stellen eingebaute Analog-Digital-Wandler zur Verfügung. Somit kann ein nichtinvasiver Stromsensor über eine *Widerstandsbrücke* direkt am Raspberry Pi angeschlossen und ausgewertet werden.

3.4 Optische Zeichenerkennung

Alle oben beschriebenen Methoden der Energieverbrauchserfassung haben einen gemeinsamen Nachteil. Die Verbrauchsmessung kann nicht ohne einen elektronischen, meist selbstgebauten Sensor erfolgen. Von diesem Nachteil ist der nachfolgend dargestellte Ansatz befreit, der auf der *optischen Zeichenerkennung (OCR)* basiert. Sämtliche Zähler, die in privaten Haushalten eingebaut werden, sehen eine durch den Menschen durchführbare Ablesemöglichkeit vor und sind mit einer analogen oder digitalen Zählerstandsanzeige ausgestattet. Um den Zählerstand abzulesen, wird auf dem Zähler eine Kamera befestigt, die in regelmäßigen Zeitabständen den Zähler fotografiert. Aus den vorbereiteten Bildern wird durch einen *Ziffernerkennungsalgorithmus* der aktuelle Zählerstand ermittelt.

Die Fast Forward AG ist ein Hersteller solcher Geräte (Fastforward, 2016), die unter dem Namen **Fast EnergyCam** (Abbildung 3.11, S. 25) vertrieben werden.

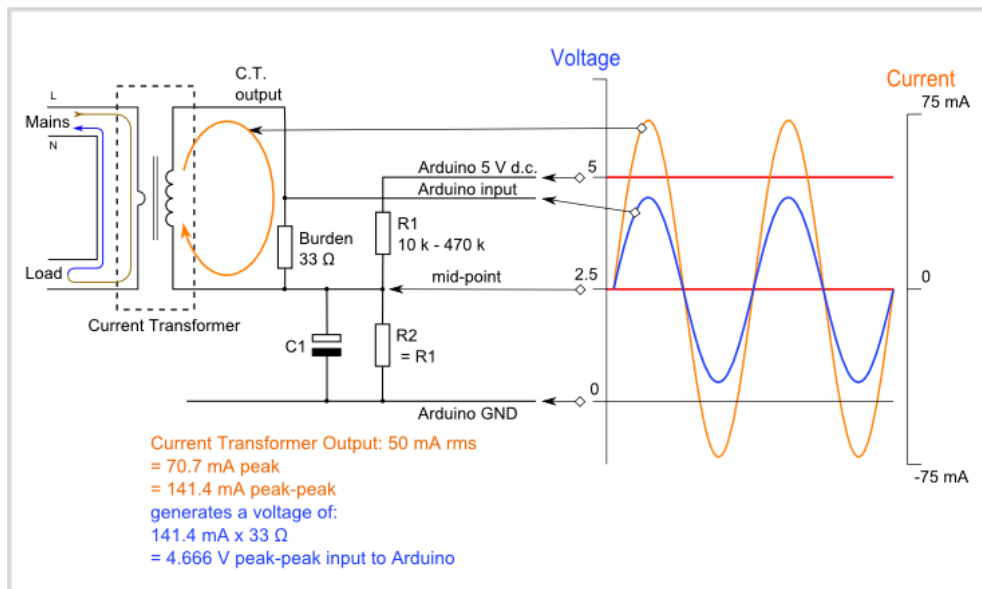


Abbildung 3.10: Anschluss eines NI-Stromsensors an einen Einplatinenrechner



Abbildung 3.11: EnergyCam von Fast Forward AG

Das Energie-Monitoring-System von Fast Forward umfasst neben einer Kamera auch ein *Gateway*, das die erfassten Daten speichert. Die Verbrauchsstatistiken können mittels einer Software visualisiert werden. Das minimale Ableseintervall beträgt bei der EnergyCam eine Minute. Des Weiteren ist der Einsatz einer proprietären Visualisierungssoftware nötig. Das System ist stark zweckgebunden und besteht aus Software- und Hardwarekomponenten eines bestimmten Herstellers. Ein *Basic-Set*, bestehend aus einer EnergyCam und einem **USB-Interface**, kann gegen einen vergleichsweise hohen Preis erworben werden. Momentan wird das System für 249 € angeboten ⁴.

In dieser Masterarbeit wird ein System entwickelt, das zur Erfassung des Zählerstands ebenso die optische Zeichenerkennung verwendet, jedoch keine solch hohen Anschaffungskosten verursacht. Im nächsten Kapitel werden alle Hardwarekomponenten des Systems aufgelistet und eine Kostenrechnung erstellt. Als Grundlage für das entwickelte System dienen zwei ähnliche Projekte, die in **Kapitel 2.4** vorgestellt wurden.

⁴<http://www.ehomeportal.de/smartmeter-energycam.htm>

Teil II

Komponenten des Systems

Kapitel 4

Hardwarekomponenten

Für die Zukunft sind Computer mit weniger als 1,5 Tonnen Gewicht vorstellbar.

POPULAR MECHANICS, 1949

An dieser Stelle werden die im Rahmen der Masterarbeit verwendeten Hardwarekomponenten vorgestellt. Neben dem Einplatinencomputer Raspberry Pi werden weitere benötigte Bestandteile beschrieben. Am Ende des Kapitels befindet sich eine Kostenübersicht aller Systembausteine.

4.1 Raspberry Pi

Wie bereits im vorherigen Kapitel erwähnt wurde, wird zur Zählerstandserfassung ein OCR-Algorithmus verwendet. Dafür müssen die Bilder in Echtzeit erfasst werden. Des Weiteren müssen aufwendige Bildervorbereitungs-, Verarbeitungs- und Ziffernerkennungsalgorithmen ausgeführt werden. Der Zählerstand muss persistent gespeichert werden und die Verbrauchsstatistiken müssen dem Benutzer in Form von Grafiken dargestellt werden. All diese Schritte können vom Einplatinenrechner Raspberry Pi problemlos bewältigt werden.

Die erste Version des Minicomputers kam im August 2011 auf den Markt. Bis Oktober 2015 wurden mehr als sieben Millionen Geräte verkauft (Upton, 2015). Raspberry Pi wurde von einer gleichnamigen britischen Stiftung mit dem Ziel entwickelt, jungen Menschen den Erwerb von Programmier- und Hardwarekenntnissen zu erleichtern. Die Entwicklung wurde mit mehreren Auszeichnungen bedacht. Für diverse Einsatzmöglichkeiten des Einplatinencomputers existiert ein großes Zubehör- und Softwareangebot. Es sind diverse Projekte auf Basis von Raspberry Pi bekannt. Zum Beispiel kann der Minirechner als ein Media-Server oder sogar als die Kernkomponente eines selbstgebauten Smartphones oder einer Mobilfunkbasisstation verwendet werden.



Abbildung 4.1: Raspberry Pi Version 2 Model B

Die in dieser Masterarbeit verwendete Version des Raspberry Pi (*Version 2 Model B*, Abbildung 4.1) hat geringe Abmessungen (93 x 63,5 x 20 mm) bei einem Gewicht von 40 Gramm und ist mit einem 4-Kern-ARM-Prozessor mit einer Taktfrequenz von 700 MHz ausgestattet. Ähnliche ARM-Prozessoren werden unter anderem in modernen Smartphones verbaut. Es werden neben einem 512 MiB großen Arbeitsspeicher auch unterschiedliche Schnittstellen zur Verfügung gestellt. Für diese Masterarbeit sind vor allem die USB- und die Kamera-Schnittstelle relevant. An einem von insgesamt vier vorhandenen USB-Anschlüssen wird ein WLAN-Stick betrieben, sodass dem Minirechner ein permanenter Netzzugang ermöglicht wird. Die aktuelle *Version 3* des Raspberry Pi verfügt über einen eingebauten WLAN- sowie einen Bluetooth-Adapter. Somit wird kein separater WLAN-Stick benötigt.

4.2 RaspiCam und weitere Teile

Die Bilder werden mit einem speziell für den Raspberry Pi angefertigten Kameramodul namens **RaspiCam** (Abbildung 4.2, S. 31) aufgenommen. Das Modul wird durch ein Flachbandkabel mit dem Raspberry Pi verbunden. Selbstverständlich kann anstatt der RaspiCam eine handelsübliche USB-Webkamera verwendet werden.

Das Betriebssystem und alle Daten werden auf einer MicroSD-Speicherkarte abgelegt. Als Betriebssystem wird die Linux-Distribution **Raspbian** verwendet, die auf Debian basiert und auf die Hardware von Raspberry Pi angepasst ist. Über ein Netzteil wird der Minirechner mit Strom versorgt. Die maximale Leistungsaufnahme beträgt dabei ca. 4 Watt. Im 24-Stunden-Dauerbetrieb unter Volllast verursacht dies an einem Netzteil mit 75 %-Wirkungsgrad und angenommenen Stromkosten von 28 ct/kWh knapp 11 € Stromkosten im Jahr.

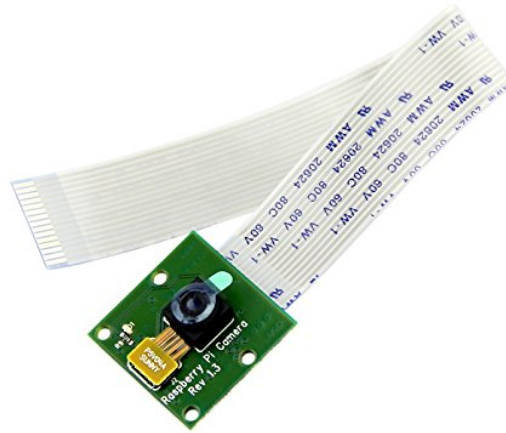


Abbildung 4.2: Kameramodul für Raspberry Pi

Komponenten	Kosten
Raspberry Pi (Version 3, Model B)	42,- €
Kameramodul	14,99 €
Netzteil	6,99 €
Gehäuse für Raspberry Pi	6,29 €
MicroSD-Speicherkarte	ab 8,- €
Kamerabeleuchtung	1,49 €
Gehäuse für Kamera und Beleuchtung	ca. 4,- €
Gesamt	83,76 €

Tabelle 4.1: Kosten des Systems

Zu den weiteren Komponenten, die für den Aufbau des Energie-Monitoring-Systems benötigt werden, zählen ein Plastikgehäuse für den Raspberry Pi sowie ein Kameramodulgehäuse. Die Energie- und Wasserzähler befinden sich meistens in Räumen mit schlechten Lichtverhältnissen, deshalb wird eine Beleuchtung benötigt, die im einfachsten Fall aus einer Leuchtdiode mit einem Vorwiderstand besteht. Die Beleuchtung steht dabei nicht ständig unter Spannung, sondern wird bei Bedarf über einen Pin des Raspberry Pi eingeschaltet.

In Tabelle 4.1 sind alle Komponenten des Systems mit deren Bezugspreisen aufgelistet. Es wird ersichtlich, dass das in dieser Masterarbeit entwickelte System im Vergleich zu den vorher betrachteten Lösungen deutlich geringere Anschaffungskosten verursacht. Im folgenden Abschnitt befindet sich eine detaillierte Beschreibung des Systemaufbaus.

4.3 Aufbau des Systems

Auf der Abbildung 4.3 ist der Aufbau des Systems dargestellt. Der Minicomputer befindet sich in einem transparenten Plastikgehäuse. Im Gehäuse sind diverse Öffnungen vorgesehen. Damit können am Raspberry Pi alle Peripheriegeräte problemlos angeschlossen werden. Der Minirechner ist an der Wand befestigt. Dafür ist an der hinteren Seite des Gehäuses eine Halterung vorhanden. Das Gehäuse wird an zwei Schrauben oder Nägeln aufgehängt und kann jederzeit abgenommen werden. Über ein Micro-USB-Kabel wird der Minirechner mit Strom versorgt. Eine Steckdose für das Netzteil befindet sich unter dem Gaszähler. In einem der USB-Ports des Raspberry Pi ist ein WLAN-Adapter eingesteckt.



Abbildung 4.3: Aufbau des Systems

Das Kameramodul ist durch ein Flachkabel mit dem Minirechner verbunden. Über zwei weitere Kabel wird die Beleuchtung eingeschaltet. Aus einem Abschnitt eines Montagelochbands ist eine Schelle angefertigt. Die Schelle ist an der oberen Befestigungsmutter

des Gaszählers angebracht und wird mit einer passenden Schraube mit einer Flügelmutter gespannt. An der Schelle ist ein Ende von der flexiblen Halterung befestigt. Das andere Ende der Halterung ist mit einem Plastikgehäuse verbunden, in dem sich das Kameramodul und die Beleuchtung befinden. Die Verwendung einer flexiblen Halterung erleichtert das Ausrichten der Kamera. Im Dauerbetrieb hat sich jedoch herausgestellt, dass sich die Kamera leicht verstellen lässt und neu justiert werden muss. Daher soll anstatt einer flexiblen eine starre Halterung verwendet werden. Diese kann zum Beispiel aus vierkantigen Aluminiumrohren angefertigt werden. Das Kameragehäuse ist so befestigt, dass das Zählwerk nicht verdeckt ist und jederzeit abgelesen werden kann. Außerdem darf die Beleuchtung nicht direkt auf das Zählwerk ausgerichtet werden, damit Lichtreflexionen auf der Abdeckscheibe vermieden werden.



Abbildung 4.4: Kameragehäuse



Abbildung 4.5: Leselampe

Das Kameramodul und die Beleuchtung sind in einem Modul-Gehäuse mit den Abmessungen $53 \times 37 \times 21$ mm untergebracht (Abbildung 4.4). Im Gehäuse wurde eine runde Öffnung für das Kameraobjektiv und eine rechteckige Öffnung für die Linse der Beleuchtung angefertigt. Des Weiteren wurde an der unteren Gehäusesseite eine schmale Öffnung für das Flachkabel und die Beleuchtungskabel vorgesehen. Bevor das Raspberry-Pi-Kameramodul eingesetzt werden kann, muss die Brennweite des Objektivs eingestellt werden. Werksseitig ist die Objektivverschraubung mit Kunststoffkleber fixiert. Um den Kleber zu lösen, muss das Objektiv mit einem geeigneten Werkzeug vorsichtig gedreht werden. Schließlich muss die Brennweite so eingestellt werden, dass die Kamera scharfe Zählerbilder aufnimmt. Es können Fehler der Bilderverarbeitung auftreten, wenn die Schärfe des Bildes nicht ausreichend ist.

Für die Beleuchtung wurde eine günstige Leselampe verwendet (Abbildung 4.5). Nach der Demontage des Lampengehäuses wurden die LED samt Verkabelung und Linse im Kameragehäuse montiert. Das Kameramodul, die LED und die Linse sind mit dem Schmelz-

klebstoff im Kameragehäuse fixiert. In der rechten Seite des Kameragehäuses ist eine runde Öffnung für die flexible Halterung vorgesehen.

Die Hardwarekomponente des Systems kann auch anders aufgebaut werden. Anstatt eines Kameramoduls kann eine Webkamera mit integrierter Beleuchtung eingesetzt werden. Dementsprechend muss eine passende, stabile Halterung angefertigt werden. Jedoch hat sich die in diesem Abschnitt vorgestellte Lösung im Dauerbetrieb bewährt und wird zum Nachbau empfohlen. Im nächsten Kapitel wird die Implementierung der Softwarekomponente des Systems beschrieben.

Teil III

Implementierung

Kapitel 5

Implementierung der Softwarekomponente

*Software ist ein Gas: Sie dehnt sich aus,
bis sie ihren Behälter voll ausfüllt.*

NATHAN MYHRVOLD – EX-TECHNOLOGIECHEF VON MICROSOFT

Nachfolgend wird die Implementierung des Softwareteils des Energie-Monitoring-Systems dargestellt. Im ersten Abschnitt werden die verwendeten Software-Bibliotheken beschrieben. Das zweite Unterkapitel behandelt den Aufbau der gesamten Software. In den weiteren Abschnitten wird auf die Umsetzung der einzelnen Softwarekomponenten im Detail eingegangen.

5.1 Eingesetzte Software-Bibliotheken

Für die Implementierung der Software wurde die *interpretierte* Programmiersprache *Python* gewählt (Lutz, 2013). Im Vergleich zum Programmcode anderer Programmiersprachen ist der Python-Code teilweise deutlich kürzer und besser lesbar (Python, 2016). Dadurch wird die Produktivität von Entwicklern um ein Vielfaches gesteigert, verglichen mit kompilierten oder statisch typisierten Sprachen wie C, C++ oder Java (Lutz, 2013). Python unterstützt objektorientierte Programmierparadigma. In seiner Standardbibliothek ist eine große Sammlung vorgefertigter Funktionalität enthalten. Diese Funktionalität kann mittels diverser Bibliotheken von Drittanbietern erweitert werden. Python wird sowohl von einer riesigen Anzahl an Privatbenutzern als auch von vielen namhaften Unternehmen, zu den unter anderem Google, Yahoo!, IBM und Hewlett-Packard gehören, eingesetzt. Denkbar sind dabei alle möglichen Einsatzszenarien: Programmierung von Internet-Diensten, Erstellung von Hardware-Tests, GUI- und Datenbank-Programmierung, Entwicklung von Computerspielen und Lösen von Aufgaben künstlicher Intelligenz.

Python ist kostenlose Open-Source-Software und standardmäßig in vielen Unix-Distributionen enthalten. Wie bereits oben angedeutet wurde, ist Python die interpretierte Programmiersprache. Das heißt, dass für das Ausführen des Programmcodes ein *Interpreter* zuständig ist. Die Python-Anweisungen, die sich in einer Textdatei mit der Erweiterung `.py` befinden, werden vom Interpreter gelesen und in den *Byte-Code* überführt. Der Byte-Code wird schließlich von der *virtuellen Maschine (PVM)* ausgeführt. Wenn der Interpreter über Dateien-Schreibrechte verfügt, wird der kompilierte Code in einer Textdatei mit der Erweiterung `.pyc` abgelegt. Dadurch wird die Programmausführung beschleunigt. Der kompilierte Code und die PVM können in einer ausführbaren Datei (Erweiterung `.exe` unter Windows) zusammengefasst werden.

Für die Bilderverarbeitung wurde die freie Programmierbibliothek *OpenCV (Open Source Computer Vision)* eingesetzt (OpenCV, 2016e). Die Bibliothek ist in der Programmiersprache C/C++ implementiert und verfügt über C-, C++-, Python- und Java-Programmierschnittstellen (Bradski, 2008). Die meisten Betriebssysteme (Windows, Linux, Mac OS, iOS und Android) werden von OpenCV unterstützt. Die Bibliothek ist in mehrere Funktionsmodule unterteilt, die über 2500 optimierte Algorithmen für Bildbearbeitung enthalten (OpenCV, 2016e). Die Algorithmen können für diverse Zwecke benutzt werden: Gesichtserkennung, Identifizierung und Klassifizierung von Objekten, Bewegungserkennung, Merkmalsextraktion und Mustererkennung usw. Genauso wie Python wird OpenCV von vielen Unternehmen (Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda und Toyota) für die Bewältigung unterschiedlicher Aufgaben verwendet. Zu der Bibliothek existiert eine ausführliche Dokumentation (OpenCV, 2016b), eine große Anzahl an Programmierbeispielen und mehrere Lehrbücher.

Alle vom Softwaresystem generierten Daten werden in der schemafreien, *dokumentenorientierten NoSQL-Datenbank MongoDB* gespeichert. Die Datenbank ist in der Programmiersprache C++ implementiert und unter einer Open-Source-Lizenz verfügbar (MongoDB, 2016). MongoDB verwaltet *Sammlungen (Collections)* von JSON-ähnlichen Dokumenten. Eine Collection enthält mehrere Dokumente und kann mit einer Tabelle einer relationalen Datenbank verglichen werden. Die Dokumente können unterschiedlich aufgebaut sein und müssen keinem festem Schema folgen (Trelle, 2014). MongoDB wird unter anderem im *BigData*-Bereich verwendet. Die Datenbank unterstützt verschiedene Aggregationswerkzeuge, unter anderem *MapReduce* und eine Gruppierungsfunktion. Des Weiteren arbeitet MongoDB weitgehend im RAM, was zu einer hohen Leistung für Lese- und Schreibvorgänge führt (MongoDB, 2016). Dabei werden die Daten zunächst im Arbeitsspeicher abgelegt und in periodischen Abständen auf die Festplatte geschrieben. Das ergibt einen Nachteil, da bei einem Systemabsturz alle nur im RAM vorhandenen Daten verloren gehen. Bei einem Systemneustart wird die abgestürzte Datenbank nicht automatisch gestartet und muss unter Umständen repariert werden. MongoDB ist mit Treibern für C/C++, Java, Python, PHP und anderen Programmiersprachen ausgestattet. Des Weiteren ermöglicht die Datenbank eine serverseitige JavaScript-Ausführung (zum Beispiel für MapReduce, s. **Kapitel 5.3.5**). MongoDB wird unter anderem für die Internetpräsenz von MTV Networks, Foursquare, The New York Times und SourceForge verwendet.

Die Implementierung des Serverteils der Frontend-Systemkomponente (s. **Kapitel 5.4**) erfolgte mithilfe des *Web Application Frameworks Flask* (Flask, 2016c). Wie die anderen

eingesetzten Bibliotheken gehört Flask zur Open-Source-Software. Das Framework wurde in der Programmiersprache Python implementiert. Die Flask-Webanwendung kommuniziert über die *WSGI*-Schnittstelle mit einem Webserver. Für Debug- und Testzwecke kann während der Entwicklung der von Flask mitgelieferte Webserver verwendet werden. Für die meisten gängigen Funktionen, wie zum Beispiel Authentifizierung, Cookies, Internationalisierung und Datenbankanbindung, werden verschiedene Erweiterungen angeboten (Flask, 2016a).

Für die clientseitige Datenbearbeitung und Darstellung wurden zwei *JavaScript-Bibliotheken* verwendet. Viele Benutzereingaben und Serveranfragen werden mithilfe der Open-Source-Bibliothek *jQuery* (jQuery, 2016) bearbeitet. Die Bibliothek besteht aus einer JavaScript-Datei, in der alle grundlegenden Funktionen enthalten sind (Chaffer und Swedberg, 2012). Für die Diagrammerstellung wurde für interaktive Grafiken die Bibliothek *Highcharts* (Highcharts, 2016) eingesetzt. Für private Webanwendungen kann die Bibliothek kostenlos benutzt werden. Viele bekannte Unternehmen, unter anderem Facebook, Twitter, Yahoo!, Visa, Groupon und Yandex, verwenden Highcharts. Für die Bibliothek ist eine umfassende Dokumentation mit zahlreichen Programmierbeispielen vorhanden.

5.2 Systemarchitektur

In der Abbildung 5.1 ist die Architektur des Softwaresystems dargestellt. Es handelt sich um ein Softwaremodell, das unter dem Namen *3-Schichten-Architektur* bekannt ist. Die Software besteht aus drei Teilen – einer *Grafik-Schicht*, einer *Logik-Schicht* und einer *Daten-Schicht*.

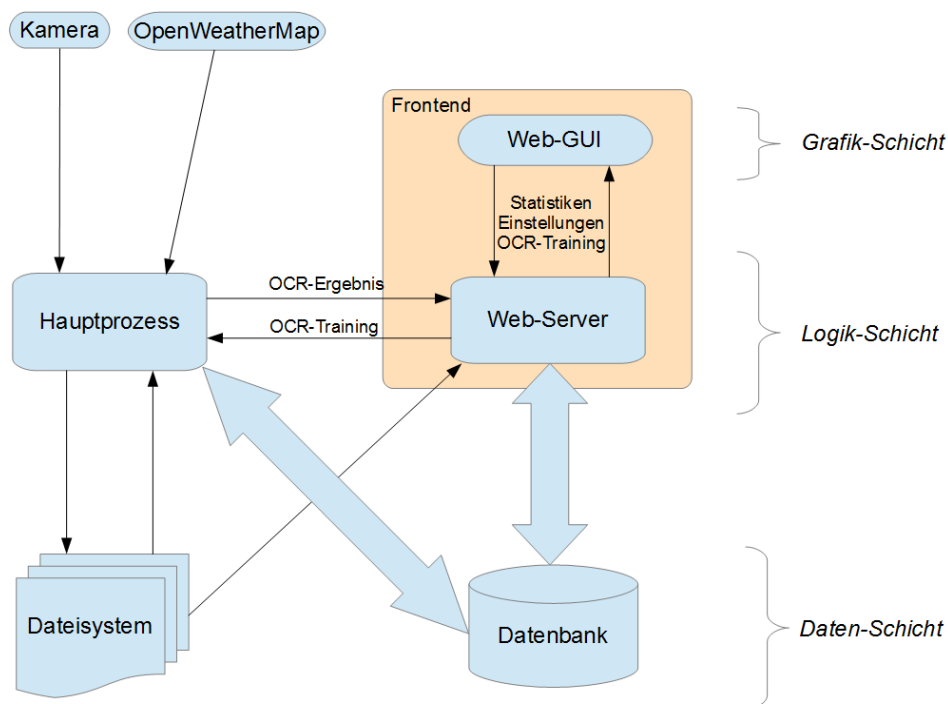


Abbildung 5.1: Systemarchitektur

Die Grafik-Schicht besteht aus einer Web-Oberfläche, die dem Benutzer in einem Webbrowser angezeigt wird. Neben der grafischen Darstellung von statistischen Daten zum Energieverbrauch hat der Benutzer die Möglichkeit, die Systemeinstellungen zu ändern und den Ziffernerkennungsalgorithmus zu trainieren. Die Benutzereingaben werden von einem Webserver entgegengenommen, verarbeitet und an andere Systemkomponenten weitergeleitet. Die grafische Benutzeroberfläche und der Webserver bilden die *Frontend-Komponente* des Systems, die in **Kapitel 5.4** behandelt wird. Alle anderen Teile des Systems gehören zur *Backend-Komponente*, die in **Kapitel 5.3** beschrieben wird.

Die Logik-Schicht umfasst neben dem bereits erwähnten Webserver einen *Hauptprozess*. Die von der Kamera erfassten Bilder werden an diese Hauptkomponente des Systems übergeben. Die Aufnahmen werden vorbereitet und verarbeitet. Dabei wird der Zählerstand durch einen OCR-Algorithmus erkannt und in einer Datenbank dauerhaft gespeichert. Außerdem werden die Bilder im Dateisystem abgelegt, damit vom Benutzer eine nachträgliche Kontrolle der Ziffernerkennung durchgeführt werden kann. Falls der Zählerstand aus einem

Grund nicht erkannt wurde, kann der Benutzer diesen manuell eingeben. Diese Eingabe wird ebenso in der Datenbank gespeichert und kann wahlweise für das Trainieren des OCR-Algorithmus verwendet werden. Am Raspberry Pi können gleichzeitig mehrere Kameras betrieben werden. Deshalb ist die Software so konzipiert, dass pro Kamera-Input ein Hauptprozess mit allen zugehörigen Einstellungen erzeugt wird. Die Einstellungen sowie Trainingsdaten des OCR-Algorithmus werden in der Datenbank persistent gespeichert. Die Datenbank bildet zusammen mit dem Dateisystem die untere Anwendungsschicht – die Daten-Schicht.

Außer Kamera-Inputs sind auch andere Eingabequellen denkbar. Der Online-Dienst *OpenWeatherMap* (OpenWeatherMap, 2016) wird ebenso als ein Input betrachtet. Des Weiteren ermöglicht die Systemarchitektur die Datenerfassung mittels diverser Sensoren. Über eine am Raspberry Pi vorhandene Schnittstelle können weitere Bauteile angeschlossen werden (zum Beispiel ein Temperatursensor oder *emonTx* – ein Modul zur Erfassung des Stromverbrauchs, s. **Kapitel 2.3**). Um diese Module betreiben zu können, muss ein entsprechender Hauptprozess implementiert werden, der über die Schnittstellen mit den restlichen Softwarekomponenten kommuniziert. In dieser Masterarbeit wird jedoch ein Hauptprozess implementiert, der für einen Kamera-Input bestimmt ist. Weiterführende Informationen zu anderen Eingabequellen sind in **Kapitel 6.2** enthalten.

5.3 Backend

In diesem Unterkapitel wird die Backend-Komponente des Systems beschrieben. Zuerst wird ein Überblick über den gesamten Programmablauf gegeben. In den weiteren Abschnitten werden die einzelnen Programmschritte erläutert.

5.3.1 Programmablauf

Wie bereits oben angesprochen wurde, wird vom Hauptprogramm für jede am Raspberry Pi angeschlossene Kamera ein separater Prozess erzeugt. Der Prozess (Abbildung 5.2, S. 42) besteht aus einer Endlosschleife, die die einzelnen Schritte der Bildverarbeitung und Ziffernerkennung enthält.

Als Erstes wird ein Bild vom Zählwerk aufgenommen. Bei der Aufnahme wird die Beleuchtung eingeschaltet. Die Abmessungen des aufgenommenen Bildes werden geändert, damit alle Bilder eine einheitliche Größe aufweisen. Das ist notwendig, weil die spätere Ziffernextraktion auf den vorgegebenen Ziffernabmessungen basiert. Anschließend wird das Bild in ein Graustufenbild konvertiert. Da die Kamera unter Umständen nicht absolut horizontal ausgerichtet ist, werden auf dem vorbereiteten Foto solche Linien gesucht, die fast horizontal ausgerichtet sind. Aus den Winkeln dieser Linien wird der Durchschnittswinkel berechnet. Das Bild wird um diesen Winkel rotiert. Nach diesem Schritt ist das Bild für das Extrahieren der Konturen vorbereitet.

Auf dem Bild werden Konturen gesucht, deren Abmessungen sich in den vorgegebenen Grenzen befinden. Die Vorgaben sind in den Einstellungen gespeichert und können jederzeit über die grafische Oberfläche angepasst werden. Die gefundenen Konturen stellen die Ziffernkandidaten dar. Aus allen Ziffernkandidaten werden durch einen Algorithmus

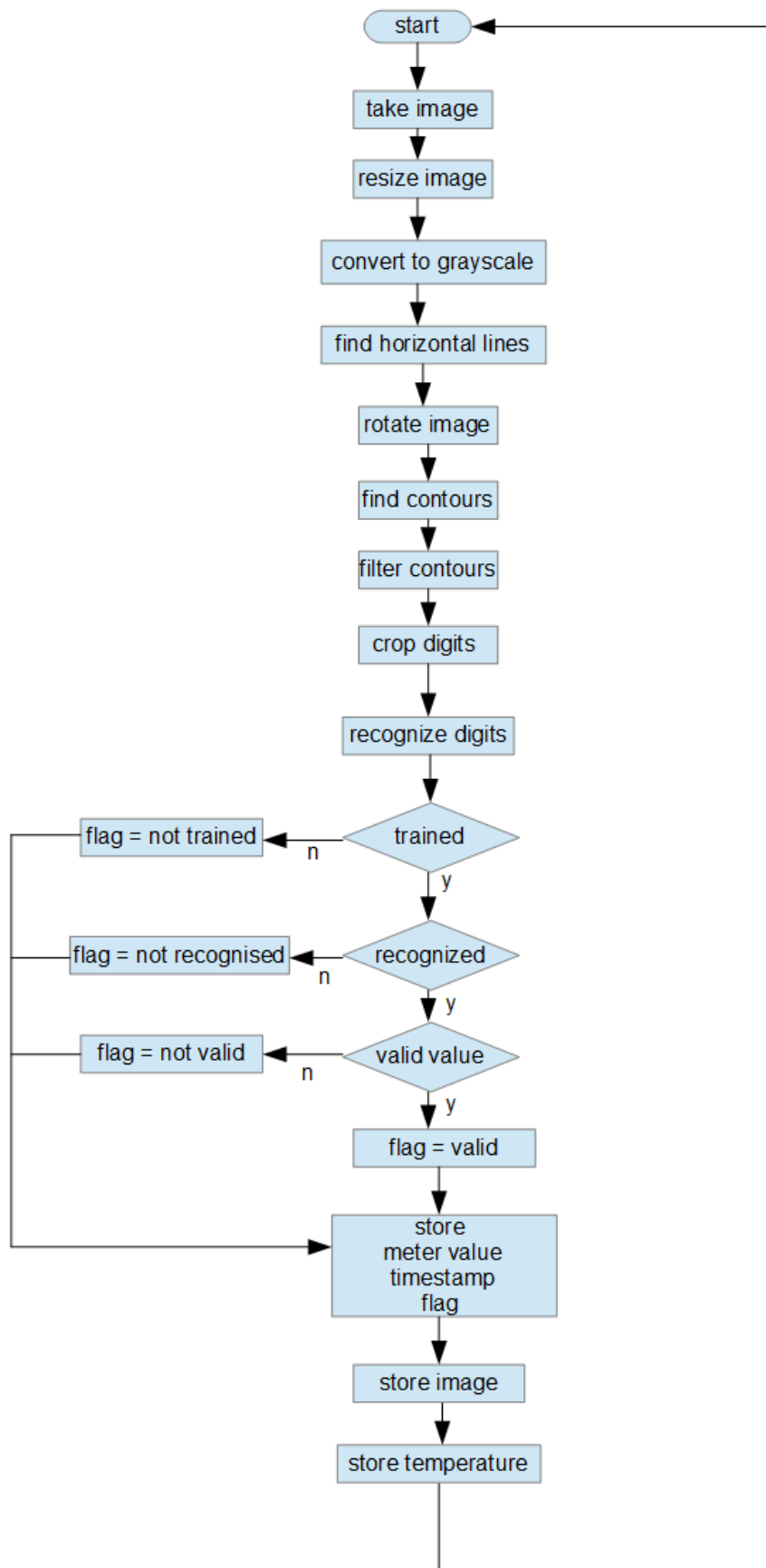


Abbildung 5.2: Programmablauf

(s. **Kapitel 5.3.3**) nur die ausgewählt, die eine Ziffer des Zählwerks repräsentieren. Die Bereiche des Zählerbilds, die die Ziffern enthalten, werden ausgeschnitten und an die Zählerstandserkennungssoftwarekomponente übergeben.

Wenn der Ziffernerkennungsalgorithmus bereits trainiert ist, wird es versucht, jedem einzelnen Ziffernbild einen numerischen Wert zuzuordnen. Der aus separaten Ziffern gebildete Zählerstand wird anschließend validiert. Falls das Algorithmus nicht trainiert ist, mindestens eine Ziffer nicht erkennen kann oder der Zählerstand nicht valide ist, so wird es in eine Statusvariable vermerkt. Zum Schluss wird der Zählerstand mit einem Zeitstempel und der Statusvariable in der Datenbank gespeichert. Das dazugehörige Bild wird im Dateisystem abgelegt. Die aktuellen Wetterdaten werden über das Internet abgerufen und in der Datenbank gespeichert. Nach einer Pause, deren Dauer vom Benutzer in den Einstellungen definiert wird, wird die Schleife von vorne ausgeführt.

5.3.2 Klassenübersicht

Die gesamte Software wurde nach dem Programmierparadigma der *Objektorientierung* modelliert und implementiert. Die dynamische Programmiersprache Python gehört zur Generation von Programmiersprachen, die die Objektorientierung unterstützen (Walerowsky, 2007). In der Abbildung 5.3 auf der nächsten Seite ist das Klassendiagramm der Backend-Komponente des Softwaresystems dargestellt. Der Übersicht halber wurden in einigen Klassen Methoden- und Variablennamen ausgelassen.

Die Hauptklasse `Raspimeter` ist von der Python-Klasse `threading.Thread` abgeleitet und umfasst zwei Objektmethoden und mehrere Klassenmethoden. Die überladene Objektmethode `run` stellt eine endlose Schleife dar. In dieser Schleife wird in periodischen Zeitabständen die zweite Objektmethode `takeAndRecognizeImage` aufgerufen. Aus dem Namen der Methode geht hervor, dass innerhalb dieser ein Bild aufgenommen und verarbeitet wird. Das erfolgt in den Schritten, die im vorherigen Kapitel erläutert wurden.

Die Klasse `Camera` implementiert einen Kamera-Input und verfügt über die einzige Methode `capture`, die als Rückgabewert ein Array liefert. Die Funktionsweise der Methode ist im nächsten **Kapitel 5.3.3** beschrieben. Jedes Bild wird intern von OpenCV als mehrdimensionales Array behandelt. Alle mathematischen und numerischen Routinen zur Bearbeitung von Arrays werden von einem mächtigen Open-Source-Python-Modul namens `NumPy` zur Verfügung gestellt (NumPy, 2016). Alle Methoden zur Verarbeitung des aufgenommenen Bildes befinden sich in der Klasse `MeterImage`, die das von der Kamera aufgenommene Bild als eine Objektvariable kapselt.

Die Klassen `Meter`, `MeterSettings` und `MeterImageSettings` stellen zusammen mit weiteren Klassen die Daten-Schicht der Software dar und werden in **Kapitel 5.3.5** beschrieben. Alle Datenbankoperationen werden mit den Methoden der Klasse `MongoDatabaseManager` ausgeführt. Für die Ziffern- und Zählerstanderkennung sind die Klassen `SingleDigitKNNRecognizer` und `MeterValueRecognizer` zuständig (s. **Kapitel 5.3.4**).

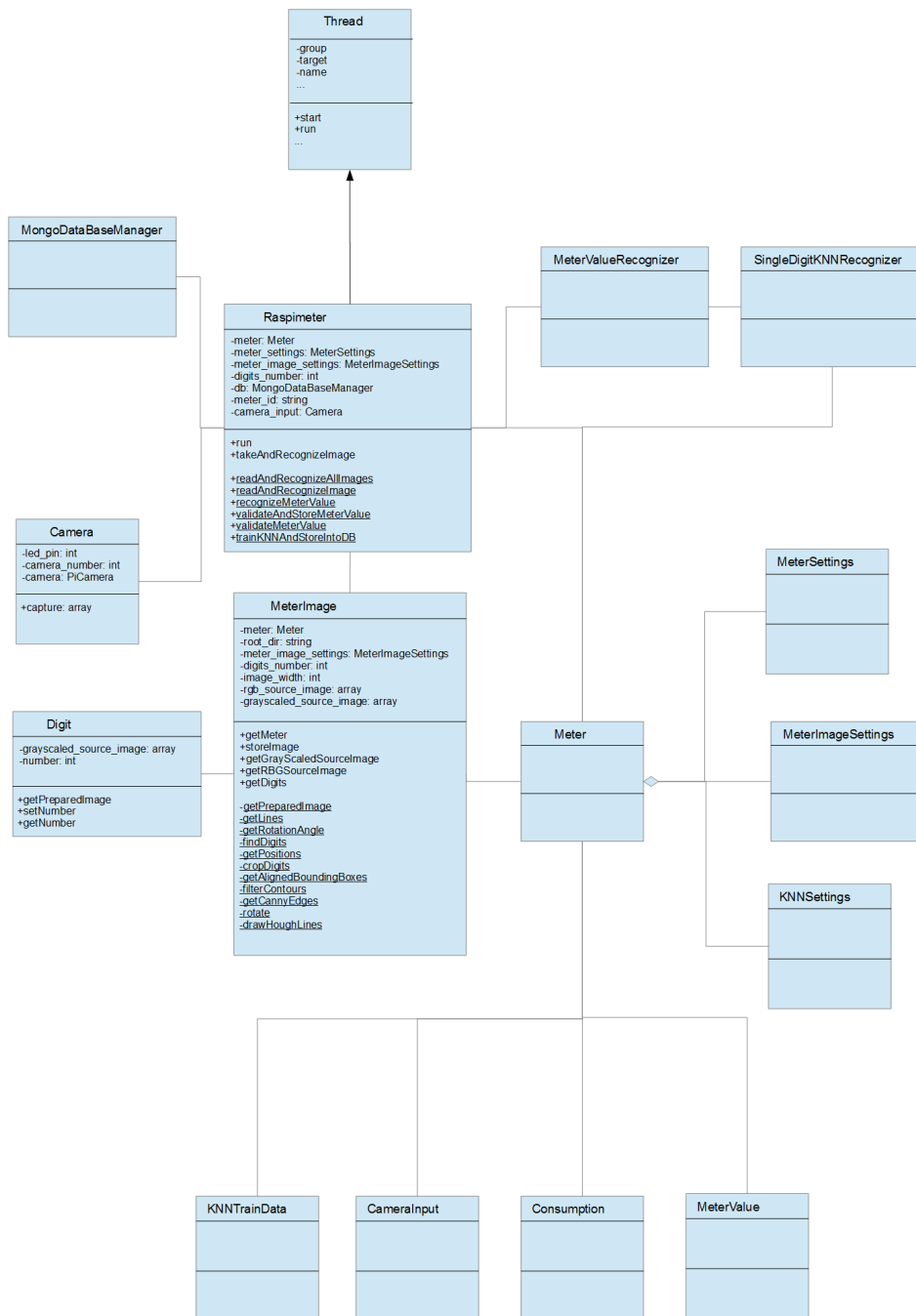


Abbildung 5.3: Klassendiagramm der Backend-Komponente

5.3.3 Bilderfassung und Vorbereitung


Alle Optionen des Raspberry-Pi-Kameramoduls sind im Python-Modul `PiCamera` implementiert. Um auf die Kamera zugreifen zu können, wird im Konstruktor der Klasse `Camera` ein `PiCamera`-Objekt instanziiert. Die `Camera`-Klasse besitzt die Methode `capture` (Abbildung 5.4). Ein Bild wird durch den Aufruf der gleichnamigen Objektmethode `capture` des `PiCamera`-Objekts aufgenommen (Zeile 5).

```

1 def capture(self):
2     GPIO.output(self.__led_pin, GPIO.HIGH)
3     rawCapture = PiRGBArray(self.__camera)
4     time.sleep(0.1) # allow the camera to warmup
5     self.__camera.capture(rawCapture, format="bgr")
6     image = rawCapture.array
7     GPIO.output(self.__led_pin, GPIO.LOW)
8
9     return image

```

Abbildung 5.4: Methode `capture` der `Camera`-Klasse



Raspberry Pi B+ B+ J8 GPIO Header			
		Pin No.	
3.3V	1	2	5V
GPIO2	3	4	5V
GPIO3	5	6	GND
GPIO4	7	8	GPIO14
GND	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10	19	20	GND
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
GND	25	26	GPIO7
DNC	27	28	DNC
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21

Abbildung 5.5: Raspberry Pi GPIO

Um eine Aufnahme unter schlechten Lichtverhältnissen zu ermöglichen, wird am Raspberry Pi eine Beleuchtung angeschlossen. Das Erde-Kabel der Beleuchtung ist mit einem der *GND-Pins* des Raspberry Pi verbunden. Das Plus-Kabel ist an einen der sogenannten *GPIO-Pins* des Raspberry Pi angeschlossen. „Die GPIO-Pins (*General Purpose Input Output*, Abbildung 5.5) bilden die zentrale Schnittstelle zwischen dem Raspberry Pi zu externen Geräten und digitalen Schaltungen. Dabei übernehmen bestimmte Pins neben

der einfachen Ansteuerung auch bestimmte Funktionen wie die Kommunikation per *I2C*, *UART* oder *SPI*“ (RaspberryPi, 2016).

Im Falle der Beleuchtung handelt es sich um eine einfache Änderung des Spannungspiegels an einem im Konstruktor definierten Pin (`led_pin`). Die Ansteuerung der GPIO-Pins ermöglicht das Python-Modul `Rpi.GPIO`. Bevor die eigentliche Aufnahme erfolgt, wird am `led_pin` die Spannung angelegt (Zeile 2). In Zeile 3 wird ein dreidimensionales `PiRGBArray` instanziiert, das die Ausgabe des Kameramoduls enthält. Nach einer kurzen Pause, während der sich das Kameramodul aufwärmt, findet die Aufnahme statt. Anschließend wird die Beleuchtung ausgeschaltet und das Bild wird zurückgeliefert.



Abbildung 5.6: Ausgangsbild

Im weiteren Verlauf der Bildverarbeitung wird ein Objekt der Klasse `MeterImage` erzeugt. Im Konstruktorauf-ruf wird dabei das zuvor aufgenommene Bild übergeben (Abbildung 5.6). Zu den weiteren Argumenten des Konstruktors gehört ein `Meter`-Objekt, das alle Einstellungen enthält, sowie der Pfad zum Zielordner, in dem alle Bilder abgelegt werden. Die Objektmethode `getDigits` von `MeterImage` liefert zwei Arrays zurück. In einem Array befinden sich die Bildausschnitte, die die einzelnen Ziffern des Zählwerks enthalten. Im zweiten Array sind die horizontalen Abstände zwischen den gefundenen Ziffern. Die Anzahl der gefundenen Ziffern sowie die Abstände zwischen den Ziffern werden für die Überprüfung der Ziffernextraktion benötigt, die in der Klasse `MeterValueRecogniser` stattfindet und in **Kapitel 5.3.4** beschrieben ist.

Für das menschliche Gehirn stellt die Extraktion von relevanten Informationen (acht weiße Ziffern, die sich auf schwarzem Hintergrund befinden und horizontal angeordnet sind) aus einem Zählerbild keine komplizierte Aufgabe dar. Um diese Aufgabe mit einem Computerprogramm zu lösen, muss ein Algorithmus verwendet werden. In dieser Arbeit wurde ein Algorithmus eingesetzt, welcher im Rahmen des Projekts *OpenCV Praxis: OCR für den Stromzähler* (s. **Kapitel 2.4**) entwickelt wurde.

```

1 def __init__(self, meter, source_image, root_dir):
2     self.__meter = meter
3     self.__root_dir = root_dir
4     self.__meter_image_settings = meter.meter_image_settings
5     self.__image_width = meter.meter_image_settings.image_width
6     self.__rgb_source_image = source_image
7     self.__grayscaled_source_image = self.__getPreparedImage(source_image.copy())
8     self.__digits_number = meter.meter_settings.digits_number

```

Abbildung 5.7: `MeterImage`-Konstruktor

Bevor die eigentliche Ziffernextraktion beginnen kann, werden im Konstruktor (Abbildung 5.7) von `MeterImage` alle Zählereinstellungen eingelesen und das Bild wird vorbereitet. Da die Ziffernabmessungen in den Einstellungen vorgegeben sind, müssen die Abmessungen des Ausgangsbilds geändert werden. Des Weiteren muss das Bild in ein Graustufenbild konvertiert werden, da sowohl für die Ziffernextraktion als auch -erkennung

ein schwarz-weißes Bild verwendet wird. Die Pixel-Farbinformationen sind für beide Algorithmen irrelevant. Außerdem verbraucht ein Farbfoto das Dreifache an Speicherplatz im Vergleich zu einem Graustufenbild. Dementsprechend wird für die Verarbeitung eines farbigen Bildes mehr Prozessorleistung und Hauptspeicher benötigt. Das Ändern der Abmessungen und die Umwandlung in ein Graustufenbild erfolgen in der privaten Objektmethode `getPreparedImage` (Abbildung 5.8). Nach dem Aufruf der OpenCV-Methode `cvtColor` (Zeile 4) erfolgt der Aufruf der Methode `resize` (Zeile 7). Zum Schluss wird mit der Methode `GaussianBlur` eine Weichzeichnung des Bildes erstellt (Szeliski, 2010). Dadurch wird der Kontrast des Bildes verringert, damit eine spätere Bildbearbeitung möglichst fehlerfrei erfolgen kann. Das vorbereitete Bild wird in der privaten Instanzvariable `grayscaled_source_image` abgelegt.

```

1 def __getPreparedImage(self, image):
2     prepared_image = image
3     if len(image.shape) == 3:
4         prepared_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
5
6     height, width = prepared_image.shape[:2]
7     prepared_image = cv2.resize(prepared_image,
8                                (self.__image_width,
9                                 int(self.__image_width * height / width)))
10    # remove noise
11    prepared_image = cv2.GaussianBlur(prepared_image, (3, 3), 0)
12
13    return prepared_image

```

Abbildung 5.8: Methode `getPreparedImage` von `MeterImage`

```

1 def __rotate(self, angle):
2     rows, cols = self.__grayscaled_source_image.shape
3     M = cv2.getRotationMatrix2D((cols/2, rows/2), angle, 1)
4     self.__grayscaled_source_image = cv2.warpAffine(
5         self.__grayscaled_source_image, M, (cols, rows))

```

Abbildung 5.9: Methode `rotate` von `MeterImage`

Damit sich alle Ziffernkonturen auf einer waagerechten Linie befinden, was ihre Identifikation erleichtert, wird das Bild rotiert. Die Rotation wird in der Methode `rotate` vorgenommen (Abbildung 5.9). Dazu wird aus dem Rotationswinkel und dem Drehpunkt eine *Rotationsmatrix* gebildet (Zeile 3). Als Drehpunkt fungiert der Mittelpunkt des Bildes, der aus der Anzahl der Zeilen (`rows`) und Spalten (`cols`) berechnet wird. Das Bild wird mit der OpenCV-Methode `warpAffine` (Zeile 4) rotiert. Den dafür benötigten Drehwinkel liefert die Methode `getRotationAngle`. Der Winkel wird als Durchschnittswinkel der im Bild vorhandenen Linien berechnet. Dabei werden nur die Linien betrachtet, die sich um maximal 30 Grad von der waagerechten Linie unterscheiden.

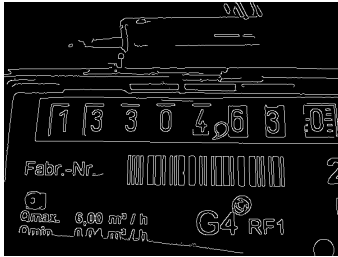


Abbildung 5.10: Mit Canny-Algorithmus gefundene Kanten

Für die Identifikation von Kanten und Linien bietet OpenCV eine große Anzahl an Methoden (OpenCV, 2016c). Eine in vielen Situationen gängige Methode zur Kantenidentifikation ist der *Canny-Algorithmus*. Der Algorithmus arbeitet auf einem Graustufenbild, auf dem die Kanten durch große Helligkeitsschwankungen zwischen zwei benachbarten Pixeln charakterisiert sind. Der Canny-Algorithmus liefert ein Bild, welches idealerweise nur noch die Kanten des Ausgangsbilds enthält (Szeliski, 2010). Zur Kantenidentifikation wird die OpenCV-Methode `Canny` (OpenCV, 2016a) aufgerufen, die Rückgabe der Methode ist auf der Abbildung 5.10 dargestellt.

```

1 def __getCannyEdges(self):
2     canny_1 = self.__meter_image_settings.canny_1
3     canny_2 = self.__meter_image_settings.canny_2
4     edges = cv2.Canny(self.__grayscaled_source_image, canny_1, canny_2)
5
6     return edges

```

Abbildung 5.11: Methode `getCannyEdges` von `MeterImage`

Die beiden *Threshold-Parameter* der `Canny`-Methode (Abbildung 5.11, Zeile 4) sind in den Einstellungen definiert und hängen vom Kontrast des Ausgangsbilds ab. Aus den gewonnenen Kanten werden im nächsten Schritt mittels der *Hough-Transformation* (Bradski, 2008) die Linien gebildet. Dafür wird die OpenCV-Methode `HoughLines` (Abbildung 5.12, Zeile 4) aufgerufen. Der dabei verwendete Parameter `lines_threshold` ist in den Einstellungen vorgegeben. Je größer dieser Wert ist, desto länger muss die zusammenhängende Linie sein.

```

1 def __getLines(self):
2     lines_threshold = self.__meter_image_settings.hough_lines
3     edges = self.__getCannyEdges()
4     lines = cv2.HoughLines(edges, 1, np.pi/180, lines_threshold)
5
6     return lines

```

Abbildung 5.12: Methode `getLines` von `MeterImage`

```

1  def __getRotationAngle(self):
2      lines = self.__getLines()
3      horizontal_lines = list()
4      theta_min = 60 * np.pi/180;
5      theta_max = 120 * np.pi/180;
6      theta_avr = 0.0;
7      theta_deg = 0.0;
8
9      for line in lines[0]:
10         theta = line[1]
11
12         if theta > theta_min and theta < theta_max:
13             horizontal_lines.append(line)
14             theta_avr += theta;
15
16     if len(horizontal_lines) > 0:
17         theta_avr /= len(horizontal_lines)
18         theta_deg = (theta_avr / np.pi * 180) - 90
19
20     return horizontal_lines, theta_deg

```

Abbildung 5.13: Methode getRotationAngle von MeterImage



Abbildung 5.14: Horizontale Linien

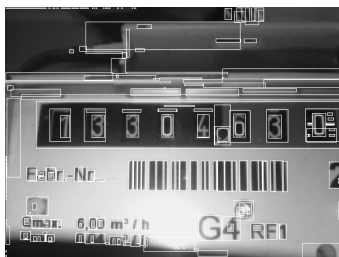


Abbildung 5.15: Alle Hüllkonturen

Bei der Methode `getRotationAngle` (Abbildung 5.13) werden zuerst aus allen Linien nur die ausgewählt, die sich um maximal 30 Grad von der waagerechten Linie unterscheiden (Zeilen 9 bis 14). Danach wird der Durchschnittswinkel gebildet und zurückgeliefert (Zeilen 16 bis 20). Das Bild ist somit für die Identifikation von Ziffern vorbereitet (Abbildung 5.14).

Die private Methode `findDigits` (Abbildung 5.16, S. 50) arbeitet mit dem Kantenbild, das der oben beschriebene Canny-Algorithmus liefert. Die Konturerkennung von OpenCV ist in der Methode `findContours` implementiert (Zeile 3).

Um jede Kontur wird in der Methode `filterContours` eine rechteckige *Hüllkontur* (*bounding box*) gezeichnet (Abbildung 5.15). Aus allen Hüllkonturen werden nur die ausgewählt, deren Abmessungen sich in den vordefinierten Grenzen befinden. Die auf diese Weise gefilterten Rechtecke werden mit darin enthaltenen Konturen zurückgeliefert (Zeile 6). Aus den gefilterten Hüllkonturen werden durch den wiederholten Aufruf der Methode `getAlignedBoundingBoxes` (Zeilen 9 bis 14) alle möglichen Kombinationen gebildet. Dabei werden die y-Positionen der Hüllkonturen ausgewertet, wodurch in jeder Kombination die größte Anzahl von Konturen bestimmt wird, die sich auf einer horizontalen Linie befinden. Die dadurch entstandene Liste `alignedBoundingBoxes` enthält mit großer Wahrscheinlichkeit die Hüllkonturen der einzel-

nen Ziffern (Abbildung 5.17). Die Methode `getPositions` (Zeile 19) liefert die nach ihrer x-Position sortierten Konturen mit den dazugehörigen Umhüllungen.

```

1 def __findDigits(self):
2     edges = self.__getCannyEdges()
3     contours, hierarchy = cv2.findContours(edges, cv2.RETR_CCOMP,
4                                           cv2.CHAIN_APPROX_SIMPLE)
5     # filter contours by bounding rect size
6     filtered_contours, boundingBoxes = self.__filterContours(contours)
7     alignedBoundingBoxes = list()
8
9     for box in boundingBoxes:
10        tmp_alignedBoundingBoxes = self.__getAlignedBoundingBoxes(box,
11                                                                    boundingBoxes)
12
13        if len(tmp_alignedBoundingBoxes) > len(alignedBoundingBoxes):
14            alignedBoundingBoxes = tmp_alignedBoundingBoxes
15
16    alignedBoundingBoxes = sorted(set(alignedBoundingBoxes))
17
18    # check for x gap and remove duplicates, calculate position
19    digits_by_position = self.__getPositions(alignedBoundingBoxes)
20    digits, x_gaps = self.__cropDigits(digits_by_position)
21
22    return digits, x_gaps

```

Abbildung 5.16: Methode `findDigits` von `MeterImage`



Abbildung 5.17: Gefilterte Hüllkonturen

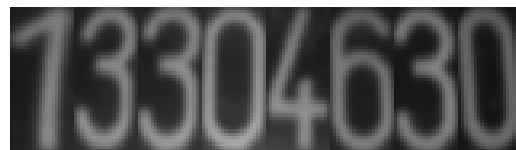


Abbildung 5.18: Gefundene Ziffern

Durch den Aufruf der Methode `cropDigits` werden aus dem Ausgangsbild die einzelnen Ziffern ausgeschnitten (Abbildung 5.18). Dabei werden die x-Abstände zwischen den Ziffern berechnet. Aus Gründen der Übersichtlichkeit ist an dieser Stelle kein Code der Methoden `filterContours`, `getAlignedBoundingBoxes`, `getPositions` und `cropDigits` enthalten. Der vollständige Quellcode befindet sich im Projekt-Repository (s. **Anhang A**).

5.3.4 Ziffernerkennung

Für die Ziffernerkennung gibt es unterschiedliche Möglichkeiten. Zum einen kann die mächtige Open-Source-OCR-Software *Tesseract* (Tesseract, 2016) eingesetzt werden. Bei einem Energie- oder Wasserzähler müssen jedoch nur zehn verschiedene Ziffern erkannt werden. Daher erscheint die Verwendung der Tesseract-Software für diese einfache Aufgabe nicht optimal, da sie für die Texterkennung bestimmt ist und außerdem viel Rechenleistung benötigt. Ein anderer Ansatz stellt das sogenannte *Template Matching* (OpenCV, 2016f) dar, das als Grundlage für den *HD_MR-Algorithmus* (*Hausdorff Distance for Meter Reading*) dient. HD_MR wurde speziell für die Erkennung von Zählerständen entwickelt (Rodríguez u. a., 2014). Eine einfachere, aber ziemlich robuste Lösung mit einer hohen Erkennungsrate

```

1 def trainAndStoreData(self, digits, responses):
2     samples = []
3     trained_responses = []
4     digits_images = []
5     new_records_count = 0
6
7     for i in range(len(digits)):
8         response = responses[i]
9
10        if response != -1:
11            samples.append(self.__reshapeImage(digits[i]))
12            trained_responses.append(response)
13            digits_images.append(digits[i])
14            new_records_count += 1
15
16        self.__knn = cv2.KNearest(np.array(samples), np.array(trained_responses))
17        self.__storeTrainedData(samples, trained_responses, digits_images)
18
19        if not self.__knn_settings.trained and new_records_count > 0:
20            self.__knn_settings.trained = True
21            self.__db.updateKNNSettings(self.__meter_id, self.__knn_settings)
22
23        return new_records_count

```

Abbildung 5.19: Methode trainAndStoreData von SingleDigitKNNRecogniser

ist die Verwendung eines Algorithmus, der auf *maschinellern Lernen* basiert. Bevor der Algorithmus in der Lage ist, eine Ziffer zu erkennen, muss er mit einer Vielzahl von Testdaten trainiert werden. Dadurch wird eine Abbildung von Ziffernbildern (*Samples*) in die dazugehörigen Ziffern (*Responses*) erstellt. In OpenCV sind viele Algorithmen für das maschinelle Lernen implementiert (OpenCV, 2016d). Eine gängige Methode der Ziffernerkennung ist der *KNN-Algorithmus* (*K-Nearest-Neighbor-Algorithmus*) (OpenCV, 2016g).

Die Ziffernerkennung erfolgt mit den Methoden der `SingleDigitKNNRecognizer`-Klasse. Das Trainieren und Speichern der Daten ist in der Methode `trainAndStoreData` implementiert (Abbildung 5.19). Beim Aufruf der Methode werden zwei Arrays übergeben. Im ersten Array befinden sich die Ziffernbilder (*Samples*). Im zweiten Array sind die Ziffern enthalten (*Responses*). Nach einer Vorbereitung wird aus den *Samples* und *Responses* durch den Aufruf der OpenCV-Methode `KNearest` ein *KNN-Modell* erzeugt. Anschließend wird das Modell in der Datenbank gespeichert. Wenn das Modell mit ausreichend Trainingsda-

```
1 def recognize(self, digit):
2     sample = self.__reshapeImage(digit)
3     test_data = np.array([sample])
4     ret, results, neighbours, dist = self.__knn.find_nearest(test_data,
5                                                             k = self.__knn_neightbors)
6
7     if neighbours[0,0] == neighbours[0,1]:
8         if dist[0,0] < self.__max_distance:
9             return int(ret)
10
11     raise SingleDigitKNNRecogniserException("Error: can't recognize digit")
```

Abbildung 5.20: Methode recognize von SingleDigitKNNRecognizer

ten trainiert wurde, ist es für die Ziffernerkennung bereit. Das Trainieren erfolgt über die Weboberfläche und ist in **Kapitel 5.4.4** beschrieben.

In der Methode `recognize` (Abbildung 5.20) wird eine einzelne Ziffer erkannt. Falls das KNN-Modell vorhanden ist, wird aus dem vorbereiteten Bild ein NumPy-Array erstellt, das als Parameter an die Methode `find_nearest` übergeben wird. Der zweite Parameter enthält die Anzahl der Nachbarn, die in den Einstellungen definiert wird. Beim Testen der in dieser Arbeit entwickelten Software wurde herausgefunden, dass die optimale Anzahl der Nachbarn gleich 3 sein soll. Wenn die Responses der zwei nächsten Nachbarn gleich sind und der Abstand zum nächsten Nachbar einen in den Einstellungen vorgegebenen maximalen Wert nicht überschreitet, dann gilt die Ziffer als erkannt. Der Rückgabewert der `find_nearest`-Methode wird zurückgeliefert. Ansonsten wird die `SingleDigitKNNRecognizerException` geworfen.

Die Erkennung des gesamten Zählerstands, der aus mehreren Ziffern besteht, findet in der Klasse `MeterValueRecognizer` statt. Die Klasse besitzt die einzige `recognize`-Methode. Beim Instanzieren eines `MeterValueRecognizer`-Objekts werden im Konstruktoraufwurf neben dem `SingleDigitKNNRecognizer` weitere Parameter übergeben. Dies sind diverse Einstellungen (die Anzahl der Ziffern, ob die letzte Ziffer ignoriert werden soll usw., s. **Kapitel 5.4.5**) und die Arrays mit den Ziffernbildern und den horizontalen Abständen zwischen den einzelnen Ziffern (Ausgabe der Methode `cropDigits`, s. **Kapitel 5.3.3**). Beim Aufruf der `recognize`-Methode werden unterschiedliche Tests ausgeführt. Wenn die Anzahl der Ziffernbilder kleiner als die tatsächliche Zifferanzahl des Zählers ist, eine oder mehrere Ziffern nicht erkannt werden können oder das KNN-Modell nicht vorhanden ist, wird dies in einer Statusvariable vermerkt. Ansonsten wird ein Array mit den erkannten Ziffern zurückgeliefert, der den aktuellen Zählerstand darstellt.

5.3.5 Datenhaltung

Wie bereits in **Kapitel 5.2** angedeutet wurde, befinden sich sämtliche Daten, mit denen die Software operiert, in einer Datenbank. Für die Datenhaltung wurde eine NoSQL-Datenbank MongoDB gewählt (s. **Kapitel 5.1**). Die Daten werden dabei in Form von Dokumenten gespeichert. Ein Dokument entspricht in etwa einer Tabellenzeile einer relationalen Datenbank. In der Abbildung 5.21 ist das *Objektdatenbankmodell* der Software dargestellt.

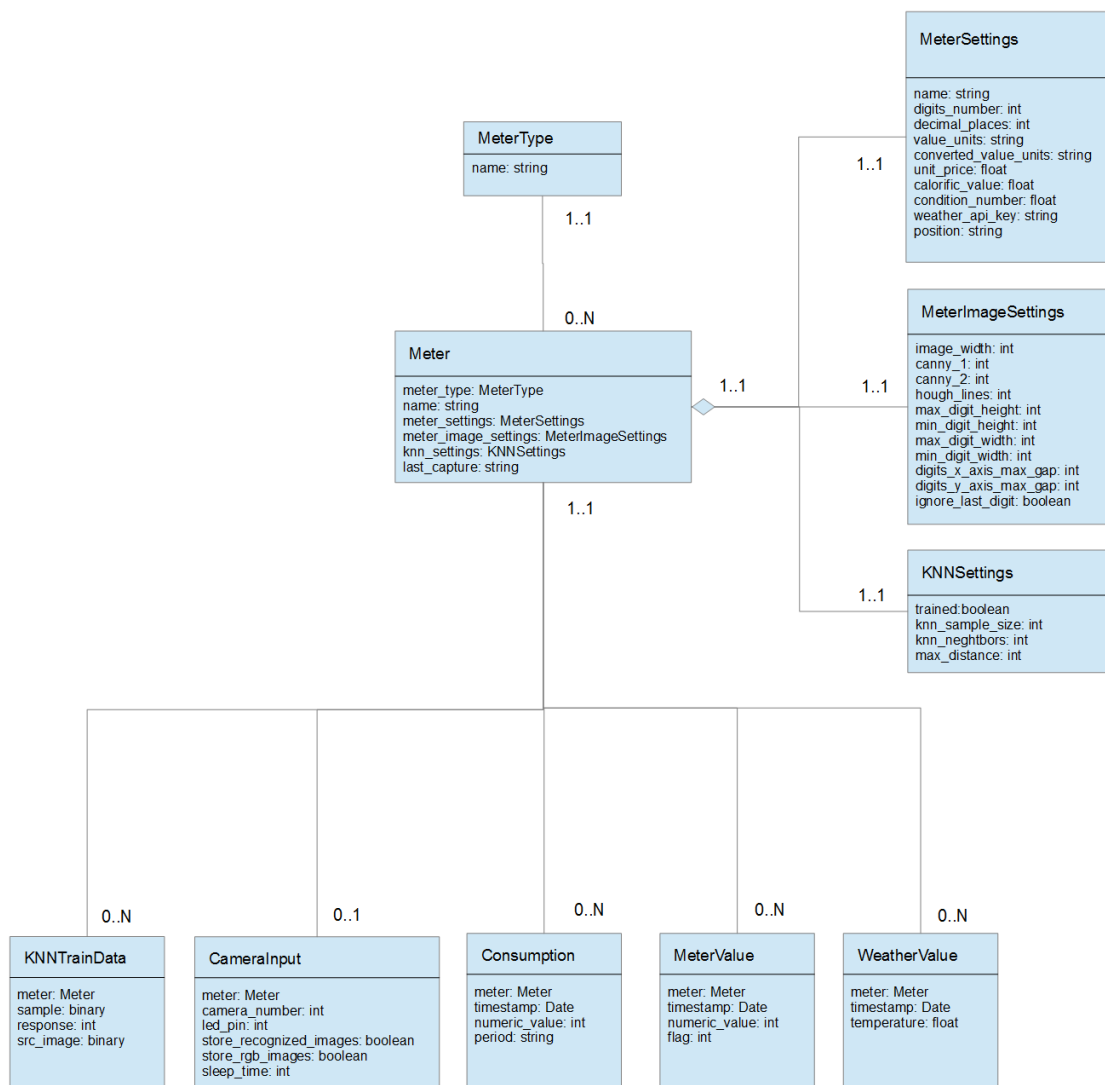


Abbildung 5.21: Datenbankmodell

Ein **Meter**-Objekt besitzt neben dem referenzierten **MeterType** noch drei Einstellungsobjekte. Die **MeterSettings**, **MeterImageSettings** und **KNNSettings** sind in das **Meter**-Objekt eingebettet. Das heißt, dass diese Objekte als *Felder* des *Meter-Dokuments* in der

Datenbank gespeichert werden. Zu einem `CameraInput` gehört genau ein `Meter`-Objekt. In der Kollektion der *MeterValue-Dokumente* sind die Zählerstände abgelegt. Die Trainingsdaten des KNN-Algorithmus, also die einzelnen Samples und Responses mit den dazugehörigen Ziffernbildern, werden als *KNNTrainData-Dokumente* gespeichert. Die Klasse `Consumption` repräsentiert den periodischen Verbrauch. Außerdem werden die Wetterdaten in Form von *WeatherValue-Dokumenten* abgelegt.

Die Abbildung der oben beschriebenen Objekte auf die Dokumente, die von MongoDB verwaltet werden, geschieht mit einem *Document Object Mapper (DOM)*. Der DOM ähnelt einer *objektrelationalen Abbildung (ORM)*, mit der ein objektorientiertes Programm seine Objekte in einer relationalen Datenbank ablegen kann. Speziell für die Programmiersprache Python und MongoDB wurde ein DOM namens *MongoEngine* (MongoEngine, 2016) entwickelt. MongoEngine ist ein Open-Source-Projekt mit einer großen Community und detaillierten Dokumentation. Außerdem stehen viele Anleitungen und Programmierbeispiele zur Verfügung.

Das Datenbankmodell befindet sich in der Datei `mongo_models.py`. Darin sind alle Klassen mit ihren Relationen abgelegt. Die Datenbankoperationen werden mit den Methoden der Klasse `MongoDataBaseManager` ausgeführt. Neben den trivialen Methoden zum Anlegen, Ändern und Auslesen von Objekten der Klasse `Meter` mit dazugehörigen Einstellungen sind hier weitere komplexe Methoden enthalten. An dieser Stelle werden einige davon erläutert.

Die Methode `storeMeterValue` wird jedes Mal aufgerufen, wenn ein Bild aufgenommen und verarbeitet wird. Wenn der Zählerstand erkannt wurde, werden die periodischen Verbräuche aktualisiert. Das geschieht in der Methode `updatePeriodicConsumptions`. Insgesamt sind fünf verschiedene Periodenlängen vorhanden (stündlich, täglich, wöchentlich, monatlich und jährlich). Für jede Periodenlänge aus dieser Liste wird die private Methode `updatePeriodicConsumption` (Abbildung 5.22, S. 55) aufgerufen.

Zuerst werden das Enddatum der Periode und das Startdatum der nächsten Periode bestimmt. Der Verbrauch der aktuellen Periode wird als Differenz zwischen dem Zählerstand am Anfang der nächsten Periode und dem Zählerstand am Anfang der aktuellen Periode berechnet. Falls in der nächsten Periode keine Zählerstände vorhanden sind, dann wird der Verbrauch als Differenz zwischen dem letzten und dem ersten Zählerstand der aktuellen Periode berechnet. Beim Erstellen der grafischen Verbrauchsstatistiken werden die vorberechneten periodischen Verbräuche und die Wetterdaten aus der Datenbank ausgelesen und an die Frontend-Komponente zurückgeliefert.

Wenn ein Zählerstand in der Datenbank gespeichert wird, werden auch die Wetterdaten aktualisiert. Die Wetterdaten werden vom Online-Dienst OpenWeatherMap über das Internet abgefragt. Der Zugriff auf die API von OpenWeatherMap erfolgt mit den Methoden des Python-Moduls `PyOWM` (PyOWM, 2016). Um die Wetterdaten abfragen zu können, wird ein *API-Schlüssel* benötigt. Dafür muss sich der Benutzer auf der Internetseite von OpenWeatherMap registrieren. Mit dem kostenlosen API-Schlüssel können sowohl die aktuellen Wetterdaten als auch eine Wettersvorhersage für die nächsten fünf Tage abgefragt werden. Der Standort kann dabei in Textform (zum Beispiel „*Osnabrueck, Germany*“) übermittelt werden. Der API-Schlüssel und der Standort werden in den Zählereinstellungen gespeichert.

```

1 def __updatePeriodicConsumption(period_start_date, period, meter):
2     period_end_date = MongoDataBaseManager.getPeriodEnd(period_start_date, period)
3     next_perion_start_date = period_end_date + timedelta(seconds=1)
4
5     try:
6         start_value = MeterValue.objects(timestamp__gte=period_start_date,
7                                           meter=meter,
8                                           flag=VALIDE_VALUE).first()
9         end_value = MeterValue.objects(timestamp__gte=next_perion_start_date,
10                                        meter=meter,
11                                        flag=VALIDE_VALUE).first()
12
13         if not end_value:
14             end_value = MeterValue.objects(meter=meter,
15                                           flag=VALIDE_VALUE,
16                                           timestamp__lte=period_end_date).
17                                           order_by('-timestamp').first()
18
19         numeric_value = end_value.numeric_value - start_value.numeric_value
20         Consumption.objects(meter=meter, timestamp=period_start_date, period=period).
21             upsert_one(set__numeric_value = numeric_value)
22
23     except Exception as e:
24         traceback.print_exc()

```

Abbildung 5.22: Methode updatePeriodicConsumption von MongoDataBaseManager

Für die Erstellung von Diagrammen werden Temperaturdaten in unterschiedlichen Auflösungen benötigt. Die Temperaturwerte werden jedoch mehrmals pro Stunde von OpenWeatherMap abgefragt und in der Datenbank gespeichert. Für jede gegebene Auflösung (Stunde, Tag, Woche, Monat und Jahr) muss aus diesen Werten ein Durchschnittswert kalkuliert werden. Die Berechnung erfolgt nach dem *MapReduce-Programmiermodell* (Miner und Shook, 2012). Das Datenbanksystem MongoDB ermöglicht das Ausführen von MapReduce-Anfragen. Die Map- und Reduce-Funktionen werden in Form von JavaScript-Funktionen als Parameter an die Methode `map_reduce` von MongoEngine übergeben. In der Abbildung 5.23 auf der nächsten Seite ist die Berechnung von stündlichen Temperaturdurchschnitten dargestellt.

Im ersten Schritt des MapReduce-Verfahrens werden mehrere Daten auf einen gemeinsamen Schlüssel abgebildet. Die Abbildung findet in der *Map-Funktion* statt (Zeilen 1 bis 8). Zu jedem Temperaturwert ist in der Datenbank ein Zeitstempel vorhanden (Zeile 3). In der Map-Funktion werden beim Zeitstempel die Sekunde und die Minute des Aufnahmedatums auf 0 gesetzt (Zeilen 4 und 5). Somit bildet die Map-Funktion auf jede volle Stunde eine Liste aus Temperaturwerten ab, die während dieser Stunde aufgezeichnet wurden. Für jede Liste wird in der *Reduce-Phase* (Zeilen 8 bis 13) ein arithmetischer Durchschnitt berechnet. Zu jedem Schlüssel liefert die Reduce-Funktion den entsprechenden durchschnittlichen Temperaturwert. Analog dazu findet die Berechnung von Durchschnitten anderer Auflösungen statt. Für die täglichen Werte wird zusätzlich die Stunde auf 0 gesetzt. Die wöchentlichen Werte werden auf den ersten Wochentag abgebildet, die monatlichen auf den ersten Tag des Monats und die jährlichen auf den ersten Tag des Jahres.

```

1 map_f = """
2     function() {
3         var key = new Date(this.timestamp);
4         key.setMinutes(0);
5         key.setSeconds(0);
6         emit(key, this.temperature);
7     }
8 """
9 reduce_f = """
10    function(key, values) {
11        return Array.sum(values) / values.length;
12    };
13 """
14 weathers = WeatherValue.objects(meter=meter, timestamp__gte=start_date,
15                               timestamp__lte=end_date)
16 weathers = weathers.map_reduce(map_f, reduce_f, {"replace": "HOURLY"})

```

Abbildung 5.23: MapReduce-Verfahren in der Methode getWeather

In Zeile 14 wird aus der Datenbank eine Kollektion von WeatherValue-Dokumenten der vorgegebenen Periode gelesen. Anschließend wird die Methode `map_reduce` mit den zuvor beschriebenen Funktionen aufgerufen (Zeile 16). Das Ergebnis ist eine Kollektion von Dokumenten, die die Durchschnittswerte in stündlicher Auflösung enthalten.

5.3.6 Autorun

```

1 from raspimeter import Raspimeter
2 from db.mongo_db_manager import MongoDataBaseManager as mdb
3
4 if __name__ == '__main__':
5     camera_inputs = mdb.getCameraInputs()
6
7     for camera_input in camera_inputs:
8         rm = Raspimeter(mdb, camera_input)
9         rm.start()

```

Abbildung 5.24: Datei runner.py

Die Backend-Komponente des Systems wird durch das Ausführen des Python-Skriptes `runner.py` gestartet. Die Datei `runner.py` enthält eine einzige `main`-Methode (Abbildung 5.24). Für jeden in der Datenbank vorhandenen `CameraInput` wird ein `Raspimeter`-Thread erstellt und gestartet. Das Kommando

```
python runner.py
```

führt das Script aus. Das Programm kann jedoch durch einen Fehler oder seitens des Betriebssystems beendet werden. Außerdem muss das Programm bei einem Neustart des Betriebssystems automatisch ausgeführt werden. Diese Aufgabe übernimmt *Supervisor* (Supervisor, 2016) – ein Kontrollsystem für Prozesse. Somit überwacht Supervisor alle von ihm gestarteten Prozesse. Falls ein Prozess beendet wurde, wird ein Prozessneustart unternom-

men. Weiterführende Informationen zur Installation und Konfigurierung von Supervisor befinden sich im **Anhang A**.

5.4 Frontend

In diesem Abschnitt wird die Frontend-Komponente des Systems beschrieben. Zuerst wird ein Überblick über die gesamte Architektur der Komponente gegeben. Nachfolgend werden die einzelnen Bestandteile des Frontends behandelt.

5.4.1 Architektur

Die Frontend-Komponente der in dieser Masterarbeit entwickelten Software ist entsprechend der *Client-Server-Architektur* implementiert. Dabei ist der Server ein Prozess, der einen Dienst anbietet. Der Client ist ebenso ein Prozess, der auf diesen Dienst zugreifen kann (Bass u. a., 2012). Für die Serverimplementierung wurde das Web Application Framework Flask verwendet (s. s. **Kapitel 5.1**). In der Abbildung 5.25 auf der nächsten Seite ist das *Sequenzdiagramm* der Frontend-Komponente dargestellt. In der Rolle des Clients tritt ein Webbrowser auf.

Die Kommunikation zwischen Client und Server erfolgt über das *Hypertext Transfer Protocol (HTTP)*. Je nach Benutzeraktion sendet der Client dem Server unterschiedliche Anfragen (*HTTP-Requests*). Der Server bearbeitet diese Anfragen und sendet eine Antwort (*HTTP-Response*). Alle Aktionen, die der Benutzer unternehmen kann, werden in den nachfolgenden Abschnitten beschrieben. Die Funktionalität der Web-Applikation ist in den Methoden implementiert, die sich in der Datei `server.py` befinden (s. **Kapitel 5.4.6**). Dabei sind unterschiedlichen *URL-Adressen* entsprechende Methoden zugeordnet, die beim Aufruf der Adresse ausgeführt werden. Insgesamt besitzt die Weboberfläche des Systems vier verschiedene Ansichten (*Views*):

- eine grafische Übersicht über die Verbrauchsstatistiken und den Temperaturverlauf (**Kapitel 5.4.2**);
- eine Liste aller im System vorhandenen Zähler mit den dazugehörigen Einstellungen (**Kapitel 5.4.3**);
- eine zu jedem Zähler gehörende Liste der aufgenommenen Bildern (**Kapitel 5.4.4**);
- eine Seite mit den allgemeinen Systemeinstellungen (**Kapitel 5.4.5**).

Beim Erstellen einer Ansicht werden eine oder mehrere Anfragen an den Server übermittelt. Der Server bearbeitet die Anfragen, führt Datenbankzugriffe aus und bereitet Daten für eine Antwort vor. Wenn der Client die Daten vom Server erhält, werden diese im Webbrowser verarbeitet. Anschließend wird die grafische Ansicht gezeichnet und dem Benutzer angezeigt.

5.4.2 Grafische Verbrauchsstatistiken

Alle Grafiken zum Energieverbrauch befinden sich auf der Startseite des Systems, die unter der URL-Adresse `http://192.168.0.100/` (*IP-Adresse* des Raspberry Pi) im lokalen Netzwerk verfügbar ist. Für jeden Zähler wird ein Chart angezeigt, das unterschiedliche Kategorien der Daten enthält:

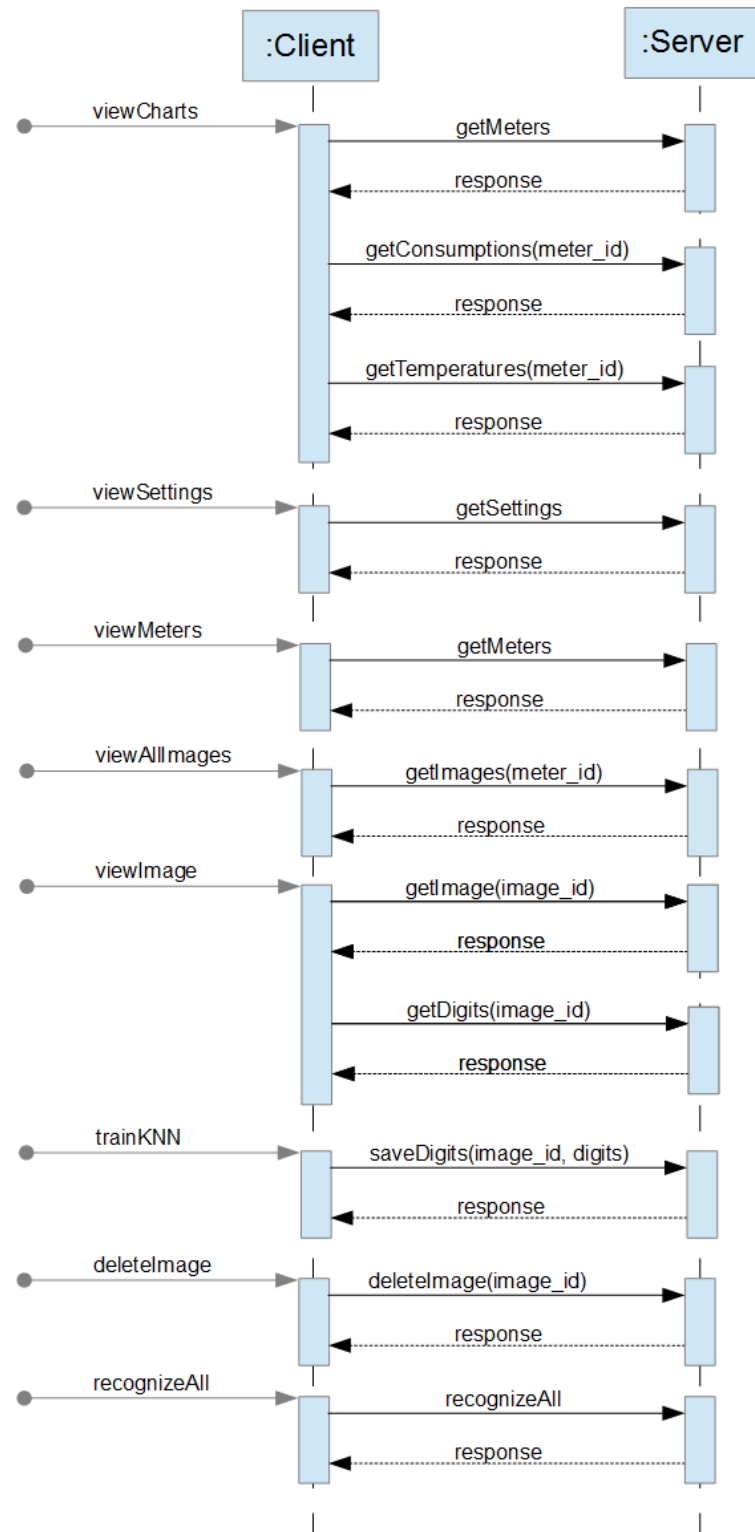


Abbildung 5.25: Sequenzdiagramm

- den Verbrauch in Zählereinheiten. Die Einheiten (zum Beispiel Kubikmeter oder Kilowattstunden) sind in den Zählereinstellungen definiert und können geändert werden.
- bei einem Gaszähler den Verbrauch in *konvertierten Einheiten*. Die in privaten Haushalten installierten Gaszähler berechnen den Gasverbrauch in Kubikmeter. Vom Energielieferanten wird jedoch nach verbrauchter Energiemenge abgerechnet, die in Kilowattstunden (kWh) ausgedrückt ist. Die Umrechnung erfolgt nach der Formel:

$$kWh = m^3 \times \text{Brennwert} \times \text{Zustandszahl}$$

Dabei ist der *Brennwert* ein Maß für die im Gas enthaltene Wärmeenergie. Dieser Wert hängt von der Qualität des Gases ab. Die *Zustandszahl* beschreibt das Verhältnis des Gasvolumens vom Norm- zum Betriebszustand. Beide Parameter können der Gasrechnung entnommen oder bei dem Energielieferanten angefragt werden. Bei einem Strom- oder Wasserzähler wird direkt nach verbrauchter Menge abgerechnet. Daher werden die konvertierten Einheiten bei diesen Zählertypen nicht angezeigt.

- für die Anzeige der Energiekosten wird die Menge von verbrauchten (im Fall von einem Gaszähler konvertierten) Einheiten mit dem Preis für eine Einheit multipliziert. Der Preis wird in den Zählereinstellungen gespeichert.
- den Temperaturverlauf.

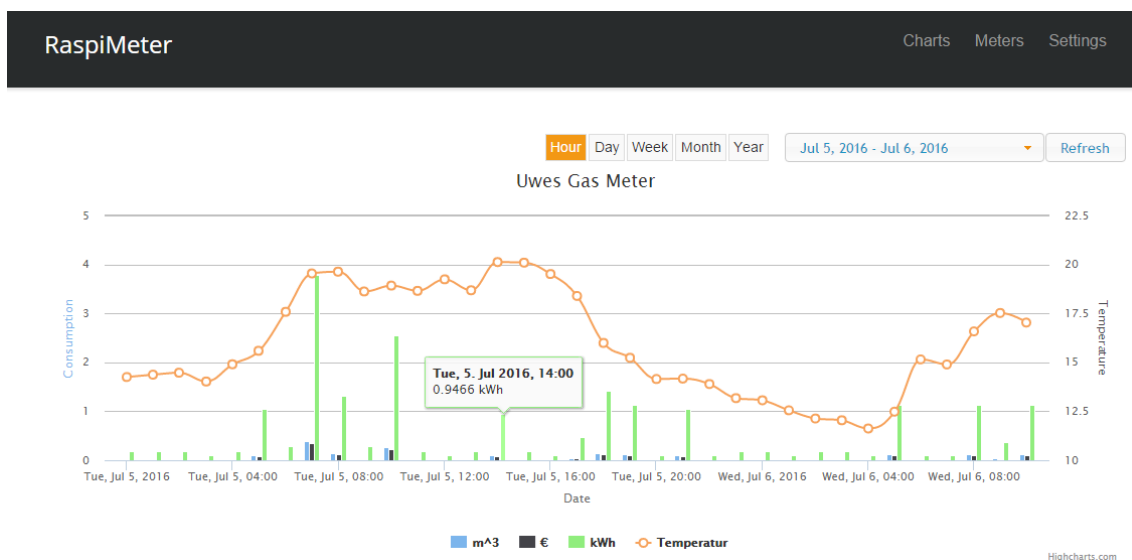


Abbildung 5.26: Grafische Verbrauchsstatistiken

Der Verbrauch und die Kosten werden in Form von Balkendiagrammen dargestellt. Der Temperaturverlauf wird als eine Linie gezeichnet (Abbildung 5.26).

Der Benutzer hat die Möglichkeit, die Grafik anzupassen. Zum einen kann das Zeitintervall beliebig gewählt werden. Dafür kann entweder das Start- und das Enddatum des

Intervalls oder eines der vordefinierten Intervalle (zum Beispiel letzter Monat) in einem *Pop-up-Fenster* gewählt werden. Zum anderen kann die Periodenlänge angepasst werden.

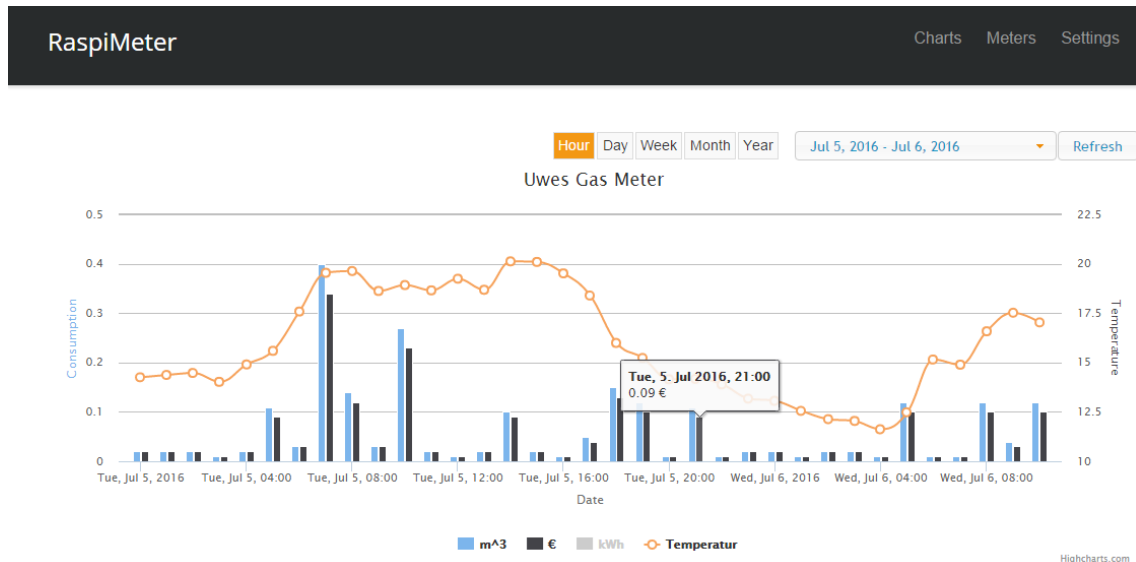


Abbildung 5.27: Anpassung von Grafiken

Standardmäßig wird der stündliche Verbrauch angezeigt. Es könnte jedoch der tägliche, wöchentliche, monatliche oder jährliche Verbrauch von Interesse sein. Um die Periodenlänge zu ändern, muss der Benutzer den entsprechenden Button anklicken. Des Weiteren werden die Grafiken automatisch aktualisiert. Das Neuladen der Daten geschieht im gleichen Turnus wie die Zählerstandsaufnahme. Außerdem kann jede Grafik durch die Betätigung des **Refresh**-Buttons manuell aktualisiert werden. Dazu muss die Legende, die sich unter der Grafik befindet, angeklickt werden. Das Chart wird automatisch skaliert und aktualisiert. In der Abbildung 5.27 ist die Anzeige der Kilowattstunden deaktiviert.

Die Charts werden im Webbrowser mit der JavaScript-Bibliothek Highcharts für interaktive Grafiken (s. **Kapitel 5.1**) gezeichnet. Alle *jQuery-Funktionen*, die im Webbrowser bearbeitet werden, befinden sich in der Datei `raspimeter.js`. Die Datei wird beim ersten Aufruf der Webseite vom Server an den Client übermittelt. Die darin enthaltenen Anweisungen werden entweder automatisch oder abhängig von den Benutzerinteraktionen ausgeführt. Beim Aufruf der Applikationsstartseite sendet der Client eine *asynchrone* HTTP-Anfrage an die URL-Adresse `http://192.168.0.100/get_meters` mittels der JQuery-Funktion `getJSON()`. Wie am Funktionsnamen ersichtlich ist, wird eine Antwort in Form eines *JSON-Objekts* erwartet. Wenn der Client eine JSON-Liste mit allen im System vorhandenen Zählern erhält, werden für jeden Zähler zwei weitere Anfragen an die URL-Adressen `http://192.168.0.100/get_consumption` und `http://.../get_weather` gesendet (Abbildung 5.28, S. 62). Als *URL-Parameter* werden dabei die Zähler-ID sowie das Start- und das Enddatum des Zeitintervalls übermittelt. Standardmäßig werden die Daten für einen Monat geladen. Sobald alle Daten vom Server empfangen wurden, wird die Funktion `renderChartContainer` aufgerufen, die das Zeichnen des Charts und des

Kontrollbuttons übernimmt. Innerhalb dieser Funktion erfolgt der Aufruf der Funktion `renderChart` (Abbildung 5.29, S. 63), in der die Grafik erstellt wird.

```

1 function loadChartDataAndRenderAllCharts(period, startMoment, endMoment) {
2   $.getJSON( "get_meters", function(meters) {
3     meters.forEach(function( meter) {
4       $.getJSON( "get_consumption?meter_id=" + meter['_id']['$oid']
5         + "&period=" + period
6         + "&start_date=" + moment(startMoment).format( DATE_FORMAT)
7         + "&end_date=" + moment(endMoment).format( DATE_FORMAT),
8
9         function(consumption) {
10          $.getJSON( "get_weather?meter_id=" + meter['_id']['$oid']
11            + "&start_date=" + moment(startMoment).format( DATE_FORMAT)
12            + "&end_date=" + moment(endMoment).format( DATE_FORMAT),
13
14            function(weather) {
15              renderChartContainer(meter, consumption, weather);
16            });
17          });
18        });
19      });
20    }

```

Abbildung 5.28: Funktion `loadChartDataAndRenderAllCharts`

Zunächst müssen die periodischen Verbräuche konvertiert werden. Die Zählerstände werden in der Datenbank als ganze Zahlen gespeichert. Daher muss der Verbrauch in eine Gleitkommazahl umgerechnet werden. Zur Umrechnung wird die Zählereinstellungsvariable `decimal_places` benötigt. Des Weiteren werden die Kosten und – falls nötig – die konvertierten Einheiten berechnet. Das Ergebnis der Funktion `calculateCostsAndConvertedConsumption` (Zeilen 3 bis 4) enthält alle Daten, die zur Erstellung der Grafik gebraucht werden. Mittels des *jQuery-Selectors* `$('#chart_' + id)` wird der *Grafikcontainer* im *Document Object Model (DOM)* der Webseite ausgewählt (Zeile 5). Der Aufruf der Funktion `highcharts` erstellt in diesem Container ein Chart. Als Erstes erfolgt die Diagrammkonfiguration (Zeilen 6 bis 13). Als Grafiküberschrift wird der Zählername verwendet. Auch die x- und y-Achsen werden mit den entsprechenden Überschriften versehen. Danach geschieht die Einstellung des Pop-up-Fensters, das auf dem Diagramm erscheint, wenn der Mauszeiger über die Grafik bewegt wird. Zum Schluss werden die Datensätze (Zählereinheiten, konvertierte Einheiten, Kosten und Temperatur) definiert. Die ersten drei Datensätze werden als vertikale Balken gezeichnet (`type: 'column'`). Die Temperaturwerte werden als ein Polynomzug (`type: 'spline'`) dargestellt. Am Ende der Funktion wird das *Parent-Element* angezeigt, in dem sich der Grafikcontainer befindet (Zeile 34).

Wenn der Benutzer die Auflösung oder das Zeitintervall ändert oder den Refresh-Button betätigt, dann wird die Funktion `redrawChart` ausgeführt. Die Funktion aktualisiert das Diagramm. Dazu werden die entsprechenden Daten für Verbräuche und Temperaturwerte vom Server geladen. Das geschieht genauso wie bei der Erstellung des Diagramms. Nachfolgend werden die Verbrauchswerte konvertiert, wie es bereits oben beschrieben wurde.

```

1 function renderChart(meter, consumption, weather) {
2   var id = meter['_id']['$oid'];
3   var costsAndConsumption
4     = calculateCostsAndConvertedConsumption(meter, consumption);
5   $('#chart_' + id).highcharts({
6     chart: {type: 'spline', zoomType: 'xy'},
7     title: {text: meter['name']},
8     xAxis: {type: 'datetime', title: {text: 'Date'}},
9     yAxis: [{title: {text: 'Consumption'},
10              {opposite: true, title: {text: 'Temperature'}}}],
11     tooltip: {headerFormat: '<b>{point.x:%a, %e. %b %Y, %H:%M}</b><br>',
12               pointFormat: '{point.y} {series.name}'},
13     plotOptions: {spline: {marker: {enabled: true}}},
14     series: [
15       {name: meter.meter_settings['value_units'],
16         data: costsAndConsumption['units'],
17         type: 'column'},
18       {name: 'Euro',
19         data: costsAndConsumption['costs'],
20         type: 'column'},
21       {name: meter.meter_settings['converted_value_units'],
22         data: costsAndConsumption['convUnits']
23         type: 'column'},
24       {name: 'Temperatur',
25         data: weather,
26         type: 'spline',
27         yAxis: 1,
28       marker:
29         {lineWidth: 2,
30          lineColor: Highcharts.getOptions().colors[3],
31          fillColor: 'white'}}
32     ]
33   });
34   $("#chart_li_" + id + ".invisible").removeClass("invisible");
35 }

```

Abbildung 5.29: Funktion renderChart

Schließlich werden die Datensätze des Diagramms erneuert. Zum Beispiel werden die Temperaturwerte mit dem Aufruf der Funktion

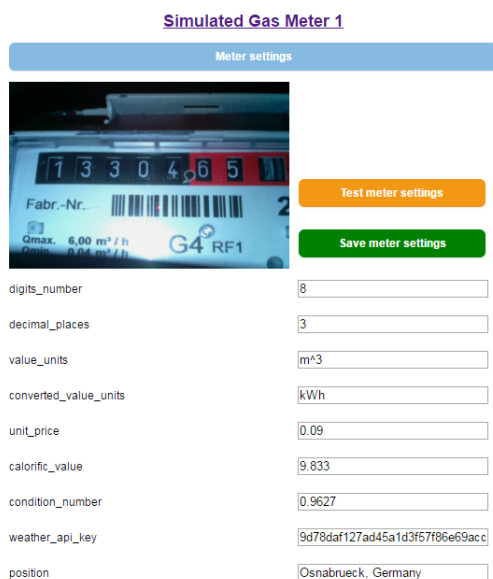
```
chart.series[3].setData(weather);
```

durch neue Daten ersetzt.

5.4.3 Zählereinstellungen

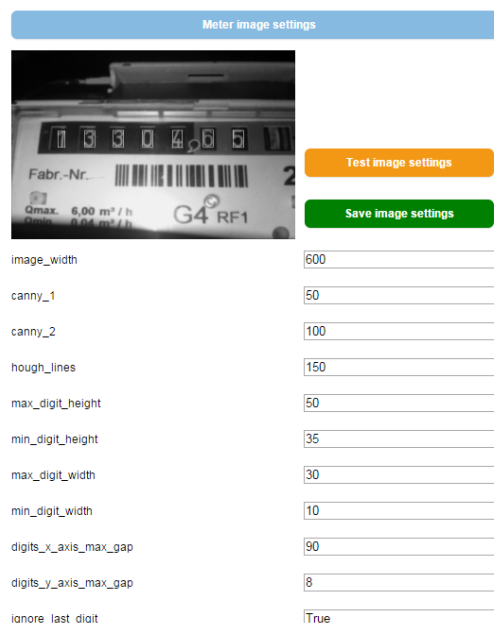
Unter der URL-Adresse `http://192.168.0.100/meters` befindet sich eine Webseite mit einer Liste aller im System vorhandenen Zähler. Für jeden Zähler sind drei Kategorien der Einstellungen vorhanden:

- die allgemeinen Zählereinstellungen (Abbildung 5.30). An dieser Stelle werden alle Variablen definiert, die für die Erstellung des Diagramms benötigt werden. Die Variable `digits_number` wird außerdem in dem Ziffernerkennungsalgorithmus verwendet. Die Parameter `decimal_places`, `value_units`, `converted_value_units`, `unit_price`, `calorific_value` und `condition_number` werden für die Umrechnung der konvertierten Einheiten und der Energiekosten gebraucht (**Kapitel 5.4.2**). Falls bei dem Zähler keine Konvertierung der Verbrauchseinheiten stattfindet (im Fall eines Strom- oder Wasserzählers), dann muss der Parameter `converted_value_units` den Wert 1.0 aufweisen. Des Weiteren werden in den Zählereinstellungen die Zugangsdaten für die OpenWeatherMap-API definiert. Darüber hinaus wird in diesem Bereich das letzte aufgenommene Bild angezeigt. Anhand dieses Bildes kann der Benutzer die Kamera optimal ausrichten.



Parameter	Value
<code>digits_number</code>	8
<code>decimal_places</code>	3
<code>value_units</code>	m ³
<code>converted_value_units</code>	kWh
<code>unit_price</code>	0.09
<code>calorific_value</code>	9.833
<code>condition_number</code>	0.9627
<code>weather_api_key</code>	9d78daf127ad45a1d3f57f8e69acc
<code>position</code>	Osnabrueck, Germany

Abbildung 5.30: Meter Settings



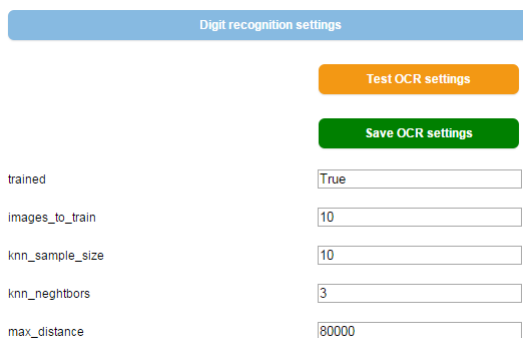
Parameter	Value
<code>image_width</code>	600
<code>canny_1</code>	50
<code>canny_2</code>	100
<code>hough_lines</code>	150
<code>max_digit_height</code>	50
<code>min_digit_height</code>	35
<code>max_digit_width</code>	30
<code>min_digit_width</code>	10
<code>digits_x_axis_max_gap</code>	90
<code>digits_y_axis_max_gap</code>	8
<code>ignore_last_digit</code>	True

Abbildung 5.31: Meter Image Settings

- die Einstellungen der Bildverarbeitung (Abbildung 5.31). Die in **Kapitel 5.3.3** beschriebenen Einstellungen des Bildverarbeitungsalgorithmus werden in der Sektion *Meter Image Settings* definiert. Alle Parameter bis auf `ignore_last_digit` werden als ganze Zahlen eingegeben. In den meisten Fällen kann die letzte Ziffer nicht erkannt werden, da die Drehgeschwindigkeit des letzten Ziffernrads am höchsten ist.

Daher ist es empfehlenswert, die letzte Ziffer zu ignorieren. Falls der Parameter `ignore_last_digit` den Wert `True` aufweist, wird bei der Zählerstanderkennung die letzte Ziffer permanent auf 0 gesetzt. Die verbrauchte Energiemenge, die durch die letzte Ziffer des Zählwerks angezeigt wird, ist zu gering (ein Liter bei einem Gaszähler mit drei Nachkommastellen). Dieser Verbrauch hat keinen Einfluss auf die grafischen Verbrauchsstatistiken, da bei der Umrechnung die Kosten und die konvertierten Einheiten auf zwei oder drei Nachkommastellen gerundet werden. Außerdem ist in dieser Sektion das Ergebnis des Bildvorbereitungsalgorithmus vorhanden. Das Bild kann zur Anpassung der Parameter benutzt werden. Wenn alle zu erkennenden Ziffern des Zählwerks mit den rechteckigen Konturen umhüllt sind, dann wurden alle Parameter optimal eingestellt.

- die Einstellungen des Ziffernerkennungsalgorithmus (Abbildung 5.32). Die boolesche Variable `trained` zeigt an, ob der Algorithmus vom Benutzer trainiert wurde. Der Parameter `knn_sample_size` gibt die Abmessungen des Ziffernbilds an, das als `sample` in der Datenbank gespeichert wird. Dieser Wert soll sich zwischen 10 und 15 befinden. Auch die Parameter `knn_neighbors` und `max_distance`, die in **Kapitel 5.3.4** beschrieben wurden, können an dieser Stelle angepasst werden.



Digit recognition settings	
	<input type="button" value="Test OCR settings"/>
	<input type="button" value="Save OCR settings"/>
trained	<input checked="" type="checkbox"/>
images_to_train	<input type="text" value="10"/>
knn_sample_size	<input type="text" value="10"/>
knn_neighbors	<input type="text" value="3"/>
max_distance	<input type="text" value="80000"/>

Abbildung 5.32: OCR Setting

Bevor die Einstellungen endgültig gespeichert werden, kann der Benutzer die Richtigkeit der Eingaben überprüfen. Dazu muss der Knopf **Test Meter Settings**, **Test Image Settings** oder **Test OCR Settings** betätigt werden. Falls keine Fehlermeldung erscheint, dann können die Einstellungen in die Datenbank geschrieben werden. Die Webseite mit den Einstellungen wird mithilfe der Bibliothek **WTFORMS** (Flask, 2016b) serverseitig generiert und an den Browser in Form eines *HTML-Dokuments* ausgeliefert.

5.4.4 Trainieren

Auf der Seite mit den Zählereinstellungen kann der Benutzer den Zählernamen anklicken. Im Webbrowser wird dadurch eine Webseite geöffnet, auf der alle aufgenommenen und gespeicherten Zählerbilder angezeigt werden (Abbildung 5.33). Die Bilder und die dazugehörigen Zählerstände werden in sieben Kategorien unterteilt. Die Zuordnung erfolgt anhand des Ergebnisses der Bilderverarbeitung oder Ziffernerkennung:

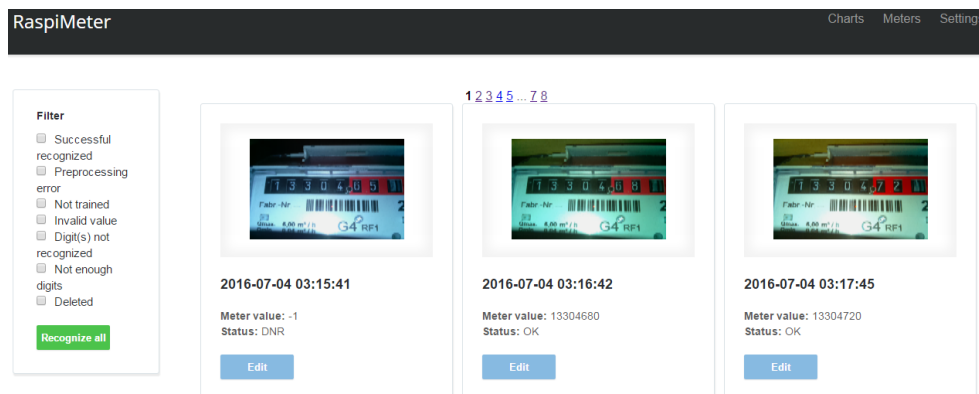


Abbildung 5.33: Zähleraufnahmen

- *Successful recognized* (Status: OK). Der Zählerstand wurde korrekt erkannt und validiert. Der Wert wird für die Diagrammerstellung verwendet.
- *Preprocessing error* (Status: PE). Bei der Vorbereitung des Bildes für die Ziffernerkennung trat ein Fehler auf. Die Einstellungen der Bildverarbeitung müssen ggf. angepasst werden.
- *Not trained* (Status: NT). Der Zählerstand wurde nicht erkannt, da der Ziffernerkennungsalgorithmus nicht trainiert ist. Der Algorithmus muss vom Benutzer trainiert werden.
- *Invalid value* (Status: IV). Der Zählerstand wurde erkannt, ist jedoch nicht valide. Das ist der Fall, wenn der erkannte Zählerstand kleiner ist als der zuletzt in der Datenbank gespeicherte Wert. Eventuell wurden einige Ziffern nicht korrekt erkannt. Der Benutzer kann den Zählerstand manuell korrigieren. Die Benutzereingabe kann für das Trainieren des OCR-Algorithmus verwendet werden.
- *Digit(s) not recognized* (Status: DNR). Eine oder mehrere Ziffern wurden nicht erkannt. Eine Benutzerinteraktion wird benötigt.
- *Not enough digits* (Status: NED). Der Bildverarbeitungsalgorithmus lieferte nicht genug Ziffern für den OCR-Algorithmus. Der Zählerstand kann manuell eingegeben werden. Die Eingabe soll jedoch für das Trainieren des OCR-Algorithmus
- *Deleted* (Status: D). Das Bild und der Zählerstand wurden als gelöscht markiert.

Bei jedem Bild sind auch weitere Informationen vorhanden. Zum einen wird das Aufnahmedatum angezeigt. Des Weiteren wird neben der Kategorie der Zählerstand dargestellt. Falls die Bildverarbeitung oder die Ziffernerkennung erfolglos war, dann wird der Zählerstand auf -1 gesetzt und bei der Erstellung des Diagramms nicht berücksichtigt. Ansonsten wird der erkannte Zählerstand als eine ganze Zahl angezeigt.

Im laufenden Betrieb werden mehrere Hunderte Bilder aufgenommen und gespeichert. Damit der Benutzer die Übersicht über die Bilder nicht verliert und der Server mit den Client Requests nicht überfordert wird, werden pro Seite maximal 20 Fotos angezeigt. Der Benutzer kann zwischen den einzelnen Seiten navigieren. Dafür wird im oberen Fensterbereich eine Seitenaufzählung dargestellt. Durch das Anklicken der Seitennummer wird die entsprechende Webseite vom Server generiert und an den Webbrowser ausgeliefert. Die serverseitige *Paginierung* ist in **Kapitel 5.4.6** beschrieben. Wenn der Benutzer ein Bild

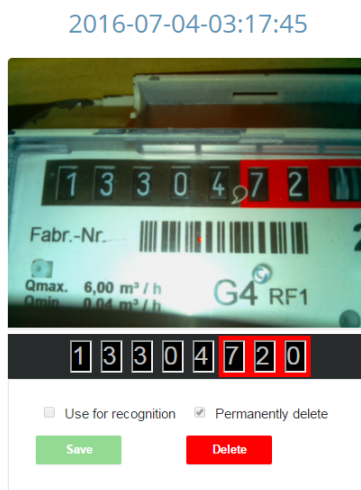


Abbildung 5.34: Korrekt erkannter Zählerstand

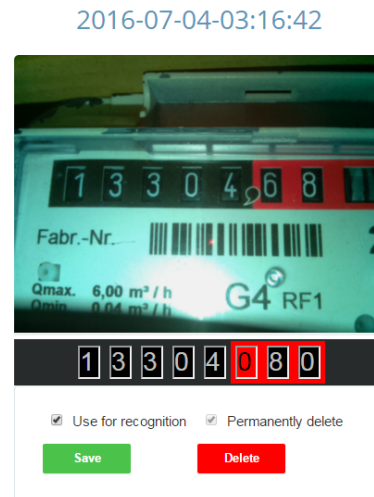


Abbildung 5.35: Nicht erkannte Ziffer

anklickt, öffnet sich ein Pop-up-Fenster. In dem Fenster befindet sich ein Zählerbild in größerer Auflösung und weitere Informationen und Eingabemöglichkeiten (Abbildung 5.34). Der Webbrowser sendet beim Anklicken eines Bildes eine HTTP-GET-Anfrage an die URL-Adresse

`http://.../get_digits?image_name=IMAGE_NAME,`

wobei `IMAGE_NAME` durch den Namen des Bildes ersetzt wird. Der Bildname besteht aus dem Aufnahmedatum, der Zähler-ID und der ID des korrespondierten Zählerstands, der in der Datenbank gespeichert wird. Der Server bearbeitet die Anfrage und startet den Bildverarbeitungs- und Ziffernerkennungsalgorithmus, wie in **Kapitel 5.3.3** bis **5.3.4** beschrieben wurde. Das Ergebnis der Ziffernerkennung wird als JSON-Dokument an den Client zurückgeliefert. In dem Dokument befindet sich ein Array mit den einzelnen Ziffern. Wenn die Erkennung ohne Erfolg beendet wurde, wird der nicht erkannten Ziffer der Wert -1 zugeordnet. Anhand dieses Arrays wird im Pop-up-Fenster eine Liste mit den Zählerzif-

fern gezeichnet. Jeder Ziffer entspricht ein einstelliges *HTML-Input-Element*. Die Eingabe ist auf numerische Werte begrenzt. Bei den nicht erkannten Ziffern ist der Hintergrund des Input-Elements rot gefärbt (Abbildung 5.35, S. 67). Somit kann der Benutzer die nicht erkannten Ziffern identifizieren und die richtige Eingabe durchführen.

Bei der Betätigung des Knopfes *Save* sendet der Client eine HTTP-GET-Anfrage an die Adresse

```
http://.../save_digits?image_name=IMAGE_NAME&responses=[1,3,3,0,4,6,8,0]
```

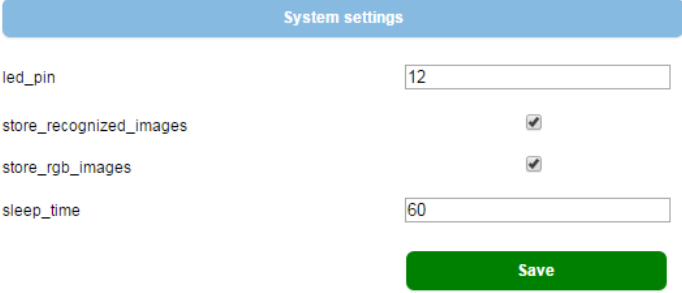
Die Benutzereingabe kann wahlweise für das Trainieren des Ziffernerkennungsalgorithmus verwendet werden. In diesem Fall muss die Checkbox *Use for recognition* ausgewählt werden. Als URL-Parameter werden der Bildname und die Liste mit den vom Benutzer eingegebenen Ziffern übergeben. Wenn das Trainieren erwünscht wurde, dann wird der Ziffernerkennungsalgorithmus mit den neuen Daten trainiert. Anschließend wird ein Versuch unternommen, den Zählerstand erneut zu erkennen. Im Fall des Erfolgs wird der erkannte Wert in die Datenbank geschrieben und das Bild wird der Kategorie *Successful recognized* zugeordnet. Ansonsten wird der vom Benutzer eingegebene Zählerstand in der Datenbank gespeichert.

Darüber hinaus kann ein Bild mit dem dazugehörigen Zählerstand gelöscht werden. Dafür muss der Benutzer den Knopf *Delete* betätigen. Falls dabei die Checkbox *Permanently delete* ausgewählt wurde, werden die Daten endgültig gelöscht. Ansonsten werden der Zählerstand und das Bild als gelöscht markiert. Die als gelöscht markierten Zählerstände werden bei der Diagrammerstellung nicht berücksichtigt.

Wenn das System zum ersten Mal in Betrieb genommen wird, ist der Ziffernerkennungsalgorithmus nicht trainiert. Die Bilder werden aufgenommen und der Kategorie *Not trained* zugeordnet. Der Benutzer soll den Algorithmus durch die manuelle Eingabe von mehreren Zählerständen trainieren. In der Praxis reichen 20 bis 30 Zählerstandeingaben aus, damit die Ziffernerkennung funktioniert. Anschließend können alle im System vorhandenen, aber noch nicht erkannten Zählerstände verarbeitet werden. Dazu muss der Knopf *Recognize all* betätigt werden. Diese massenhafte Zählerstandserkennung kann zu jedem Zeitpunkt beliebig oft wiederholt werden (zum Beispiel nach einer Anpassung der Einstellungen).

5.4.5 Systemeinstellungen

Unter der URL-Adresse `http://.../settings` ist die Webseite mit den Systemeinstellungen erreichbar (Abbildung 5.36). Auf dieser Seite wird der GPIO-Pin von Raspberry Pi definiert, an den die Kamerabeleuchtung angeschlossen ist. Der boolesche Parameter `store_recognized_images` gibt an, ob die Bilder zu den korrekt erkannten Zählerständen im Dateisystem gespeichert werden. Um die Nutzung des Speicherplatzes weiterhin zu optimieren, können die Zählerbilder als Graustufenbilder abgelegt werden. Dafür muss die Checkbox `store_rgb_images` abgewählt werden.



System settings	
led_pin	<input type="text" value="12"/>
store_recognized_images	<input checked="" type="checkbox"/>
store_rgb_images	<input checked="" type="checkbox"/>
sleep_time	<input type="text" value="60"/>
<input type="button" value="Save"/>	

Abbildung 5.36: Systemeinstellungen

Der letzte numerische Parameter gibt die Länge der Pause zwischen den Zähleraufnahmen an. Der Wert wird in Sekunden ausgedrückt. Zum einen soll die Pause nicht zu kurz sein, da ansonsten viele Bilder aufgenommen und gespeichert würden. Außerdem können die Bilder redundante Informationen enthalten. Zum anderen darf die Pause nicht allzu lang sein, da in der Praxis nicht alle Zählerstände erkannt werden. Das führt zur Entstehung von Lücken in der Verbrauchsgrafik. Da die minimale Zeitaufösung des Diagramms eine Stunde beträgt, reichen pro Stunde 10 bis 20 Zähleraufnahmen aus, um den Verbrauch korrekt zu erfassen und grafisch darzustellen.

5.4.6 Server

Die *Responses* der Serverkomponente lassen sich generell in zwei Gruppen unterteilen. Die Antworten der ersten Gruppe werden zurückgeliefert, wenn im Browser zum Beispiel eine `getJSON`-Funktion aufgerufen wurde. Der Server liefert in diesem Fall ein JSON-Dokument an den Client. Die Responses des zweiten Typs stellen HTML-Dokumente dar, die serverseitig erstellt und im Webbrowser angezeigt werden. An dieser Stelle werden zwei Servermethoden beschrieben, die jeweils zu unterschiedlichen Gruppen gehören.

Die Methode `getWeather` (Abbildung 5.37, S. 70) wird beim Aufruf der URL-Adresse `http://.../get_weather` ausgeführt. Damit der Server jeder Adresse die entsprechende Methode zuordnet, wird diese Adresse vor dem Methodennamen mit der Notation `@app.route` definiert (Zeile 1). In dieser Zeile werden auch die unterstützten *HTTP-Methods* angegeben. In den Zeilen 3 bis 6 werden aus dem `request` die Anfrage-Parameter

```

1 @app.route("/get_weather", methods=['GET'])
2 def getWeather():
3     meter_id = request.args.get('meter_id')
4     period = request.args.get('period')
5     start_date = request.args.get('start_date')
6     end_date = request.args.get('end_date')
7
8     weather = db.getWeather(period=period, meter_id=meter_id,
9                             start_date=start_date, end_date=end_date)
10
11     return json.dumps(weather)

```

Abbildung 5.37: Methode getWeather in server.py

extrahiert. Mit den Parametern erfolgt ein Aufruf der Methode `getWeather` der Klasse `MongoDataBaseManager`. Die Rückgabe dieser Methode (s. **Kapitel 5.3.5**) besteht aus einer Liste von Tupeln. In jedem Tupel befinden sich ein Zeitstempel und ein Temperaturwert. Da der Browser ein JSON-Objekt erwartet, wird die Liste in das JSON-Format konvertiert. Das geschieht mit der Methode `dumps` des Python-Moduls `json`. Das JSON-Objekt wird schließlich an den Client ausgeliefert. Die Server-Methoden `getMeter`, `getMeters`, `getConsumption`, `getImage`, `deleteImage`, `getDigits`, `saveDigits` und `recognizeAll` sind ähnlich implementiert.

```

1 @app.route("/images", methods=['GET'])
2 def renderImages():
3     meter_id = request.args.get('meter_id')
4     flag = int(request.args.get('flag'))
5     page = int(request.args.get('page'))
6
7     pagination = db.getImagesWithPagination(meter_id=meter_id, flag=flag,
8                                             page=page, per_page=20)
9     flags = ["OK", "RNV", "NT", "IV", "DNR", "NED", "D", "PE"]
10
11     return render_template('images.html', meter_id=meter_id, pagination=pagination,
12                           flag=flag, flags=flags, endpoint='renderImagesWithPagination')

```

Abbildung 5.38: Methode renderImages in server.py

In den übrigen Fällen wird dem Client ein HTML-Dokument zurückgeliefert. Die im vorherigen Abschnitt angesprochene Paginierung ist in der Methode `renderImages` implementiert (Abbildung 5.38). Wie bereits oben beschrieben wurde, werden aus der Anfrage alle Parameter extrahiert. Die Aufteilung von allen im System vorhandenen Bildern und Zählerständen übernimmt die Methode `getImagesWithPagination` der Klasse `MongoDataBaseManager`. Als Parameter erhält die Methode unter anderem die Seitennummer und die Anzahl der Objekte pro Seite. Das Ergebnis ist ein `Pagination`-Objekt, das außer den eigentlichen Zählerständen (`items`) noch weitere Attribute besitzt. Von Interesse ist die gesamte Seitenanzahl, die für die Navigation benötigt wird.

Nachdem die Daten zur Verfügung gestellt wurden, wird die Methode `render_template` aufgerufen (Zeile 11). Als Parameter erhält diese die *HTML-Schablone* (*Template*). Von

```

1 {% extends "index.html" %} {% block images %}
2 <div class="all-products page">
3   <div class=pagination>
4     {% for page in pagination.iter_pages() %}
5       {% if page %}
6         {% if page != pagination.page %}
7           <a href="{ url_for(endpoint, meter_id=meter_id, flag=flag,
8             page=page) }}">{{ page }}</a>
9         {% else %}
10          <strong>{{ page }}</strong>
11        {% endif %}
12      {% else %}
13        <span class=ellipsis>...</span>
14      {% endif %}
15    {% endfor %}
16  </div>
17  <ul class="products-list" id={{ meter_id }}>
18    {% for item in pagination.items %}
19    <li id="{{ item.timestamp.strftime('%Y-%m-%d-%H:%M:%S') }}"
20      data-index="{{ item.id }}" flag={{ item.flag }}>
21      <a href="#" class="product-photo">
22        <img class="lazy"
23          data-original="static/images/{{ 's_%s_%s_preview.png' %
24            (item.timestamp.strftime('%Y-%m-%d_%H-%M-%S'),
25              item.meter.id, item.id) }}" #
26          width="200" height="130" />
27      </a>
28      <h2><a href="#"> {{ item.timestamp }} </a></h2>
29      <ul class="product-description">
30        <li><span>Meter value: </span>{{ item.numeric_value }}</li>
31        <li><span>Status: </span>{{ flags[item.flag] }}</li>
32      </ul>
33      <button>Edit</button>
34      <div class="highlight"></div>
35    </li> {% endfor %}
36  </ul>
37 </div>
38 {% endblock %}

```

Abbildung 5.39: HTML-Schablone images.html

der *Template Engine* (Jinja, 2016) werden in diese Schablone (Abbildung 5.39) an den entsprechenden Stellen die Daten eingefügt. Dabei werden die Platzhalter (die doppelten geschweiften Klammern) mit Werten gefüllt. Innerhalb der Klammern kann der Python-Code ausgeführt werden. Die Abarbeitung von Schleifen und Abzweigungen wird von der Template-Engine ebenfalls unterstützt. Als Ergebnis entsteht ein HTML-Dokument, das im Webbrowser angezeigt wird.

Während der Implementierung der Serverkomponente wurde der Entwicklungsserver aus der Erweiterung *Flask-Script* verwendet (Flask-Script, 2016). Der Server wird durch das Ausführen des Python-Scripts `manage.py` mit dem Kommandozeilenparameter `runserver` (Abbildung 5.40, S. 72) gestartet. Als Erstes wird die oben beschriebene Webapplikation importiert (Zeile 2). In Zeile 4 wird ein `Manager`-Objekt instanziiert. Dem Objekt wird das Kommando `runserver` hinzugefügt. Das Setzen des Parameters `use_debugger` aktiviert die Ausgabe von Debug-Informationen. Wenn eine Python-Datei geändert und anschließend

```
1 from flask_script import Manager, Server
2 from flask_server.server import app
3
4 manager = Manager(app)
5 manager.add_command("runserver", Server(
6     use_debugger = True,
7     use_reloader = True,
8     host = '0.0.0.0')
9 )
10
11 if __name__ == "__main__":
12     manager.run()
```

Abbildung 5.40: Datei manage.py

gespeichert wird, soll der Server automatisch neu starten. Das wird durch das Setzen des Parameters `use_reloader` erreicht. Nach dem Ausführen des Kommandos

```
python manage.py runserver
```

ist der Server betriebsbereit. Der Server ist unter der Adresse `http://localhost:5000/` erreichbar.

Für den Dauerbetrieb wurde der *Python-WSGI-Server Gunicorn* (Gunicorn, 2016) eingesetzt. Gunicorn ist ausschließlich für unixartige Betriebssysteme bestimmt, während Flask-Server auch unter Windows betrieben werden kann. Für die Überwachung des Webserver wird für Prozesse analog zum Fall der Backendkomponente das Kontrollsystem Supervisor eingesetzt (s. **Kapitel 5.3.6**). Die Installation und Einrichtung von Gunicorn ist im **Anhang A** beschrieben.

Kapitel 6

Zusammenfassung und Ausblick

Nachfolgend werden die erreichten Ergebnisse beschrieben. Am Ende des Kapitels befindet sich ein kurzer Ausblick hinsichtlich der Erweiterungsmöglichkeiten des in dieser Arbeit entwickelten Monitoring-Systems.

6.1 Zusammenfassung

In der vorliegenden Arbeit wurde die Erstellung eines Systems für die Erfassung und Auswertung von Energieverbräuchen in privaten Haushalten beschrieben. Das System verursacht niedrigste Anschaffungskosten im Vergleich zu anderen Systemen, die sich momentan auf dem Markt befinden. Die Hardwarekomponente besteht aus Teilen, die frei verfügbar sind und von Herstellern laufend aktualisiert werden. Die Softwarekomponente des Systems ist nicht an eine bestimmte Hardwarekonfiguration gebunden. Die entwickelte Software kann auch unter diversen Betriebssystemen betrieben werden. Anstatt des Raspberry Pi kann sowohl ein anderer Einplatinenrechner als auch ein gewöhnlicher PC eingesetzt werden. Für die Implementierung der Software wurden bekannte Open-Source-Bibliotheken benutzt.

Das System verfügt über eine intuitive grafische Benutzeroberfläche. Der Aufbau des Hardwareteils und die Installation und Inbetriebnahme der Software können von einem Benutzer mit geringen IT-Vorkenntnissen erfolgreich durchgeführt werden. Das System kommt in einem privaten Haushalt seit mehreren Wochen zum Einsatz. In dieser Zeit wurden keine Ausfälle verzeichnet.

Die Erkennungsrate der einzelnen Zählerstände ist nicht allzu hoch und beträgt ca. 80 %. Das Ziel der Arbeit bestand jedoch nicht darin, einen Erkennungsalgorithmus mit möglichst hoher Erkennungsrate zu entwickeln. Die Hauptaufgabe war, die periodischen Verbräuche zu erfassen und zu analysieren. Dafür ist die oben genannte Rate an korrekt erkannten Zählerständen mehr als ausreichend. Die Aufgabe wurde somit erfolgreich gelöst.

6.2 Ausblick

Es sind noch einige offene Punkte, Verbesserungs- und Erweiterungsmöglichkeiten des Monitoring-Systems zu nennen. In der Zukunft soll das System derart erweitert werden, dass auch die Erfassung mittels anderer Sensoren möglich wäre. Es soll eine Schnittstelle zum Anschluss eines emonTx (oder einer ähnlichen Komponente) implementiert werden. Dadurch wird die Erfassung des Stromverbrauchs ermöglicht. Dafür muss der Hauptteil der Software nicht großartig verändert werden. Anstatt eines Kamera-Inputs muss ein entsprechender Input implementiert werden, der die Verbrauchsdaten des emonTx empfängt. Die Datenspeicherung und Visualisierung ist im beschriebenen System bereits vorhanden. Das wird durch die starke Modularisierung der Software ermöglicht.

Außerdem soll die grafische Weboberfläche um einige Funktionen erweitert werden. Momentan wird der Ziffernerkennungsalgorithmus anhand von kompletten Zählerständen trainiert, die aus mehreren Ziffern bestehen. Von Interesse wäre das Trainieren anhand von einzelnen Ziffern. Die nicht erkannten Ziffernbilder sollen dem Benutzer mit einer Eingabemöglichkeit angezeigt werden. Das Trainieren auf diese Art ist effizienter als das Korrigieren des gesamten Zählerstands, da meistens nur eine Ziffer nicht erkannt wird.

Des Weiteren soll die Verwaltung von Energiepreisen implementiert werden. Momentan wird der Preis für den gesamten Verbrauch in den Einstellungen gespeichert. Der Energiepreis wird jedoch seitens der Energielieferanten in periodischen Abständen (meist jährlich) angepasst. Das System soll so erweitert werden, dass jeder Periode der entsprechende Preis zugeordnet wird.

Für die Visualisierung wird aktuell eine eigene Frontend-Komponente verwendet. Um das Interesse von Teilnehmern anderer Projekte (zum Beispiel Open Energy Monitor) zu wecken, soll eine Schnittstelle implementiert werden. Die Schnittstelle soll die Integration des entwickelten Systems mit der entsprechenden Software (zum Beispiel emonCMS) ermöglichen. Das stellt allerdings keine große Herausforderung dar. Die im Rahmen anderer Projekte entwickelte Software verfügt über Schnittstellen, die den Datenimport gewährleisten.

Der komplette Quellcode des in dieser Arbeit entwickelten Systems ist in einem Git Repository unter einer Open-Source-Lizenz veröffentlicht, damit auch andere Programmierer einen Beitrag zur weiteren Entwicklung der Software leisten können.

Teil IV
Anhänge

Anhang A

Installationsanleitung

Nachfolgend wird die Installation und Inbetriebnahme des Monitoring-System beschrieben. Das Kamera-Modul soll an einem Zähler befestigt werden, wie in **Kapitel 4.3** beschrieben wurde. Es wird davon ausgegangen, dass auf Raspberry Pi das Betriebssystem Raspbian bereits installiert wurde. Der Minirechner muss an das lokale Netzwerk angeschlossen werden.

Installation von OpenCV und Python

- Die erforderlichen Entwicklertools und Pakete installieren:

```
sudo apt-get install build-essential cmake pkg-config
```
- Die notwendigen Pakete für Video- und Bild-I/O installieren. Die Pakete ermöglichen das Laden von verschiedenen Video- und Bildformaten (JPEG, PNG, TIFF usw.):

```
sudo apt-get install libjpeg8-dev libtiff4-dev libjasper-dev  
sudo apt-get install libpng12-dev libavcodec-dev libavformat-dev  
sudo apt-get install libswscale-dev libv4l-dev
```
- Die Entwicklungsbibliothek GTK installieren:

```
sudo apt-get install libgtk2.0-dev
```
- Bibliotheken installieren, die die Optimierung von OpenCV-Operationen ermöglichen:

```
sudo apt-get install libatlas-base-dev gfortran
```
- Paketverwaltungsprogramm für Python-Pakete installieren:

```
wget https://bootstrap.pypa.io/get-pip.py  
sudo python get-pip.py
```

- Python-Entwicklertools und NumPy installieren:

```
sudo apt-get install python2.7-dev
pip install numpy
```

- OpenCV herunterladen und kompilieren:

```
wget -O opencv-2.4.13.zip http://sourceforge.net/projects/opencvlibrary/
files/opencv-unix/2.4.13/opencv-2.4.13.zip/download

unzip opencv-2.4.13.zip
cd opencv-2.4.13
mkdir build
cd build

cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D
BUILD_NEW_PYTHON_SUPPORT=ON -D INSTALL_C_EXAMPLES=ON -D
INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON ..

make
sudo make install
sudo ldconfig
```

Nach diesen Schritten sind OpenCV und Python auf Raspberry Pi erfolgreich installiert.

Installation von Flask

- Die erforderlichen Python-Pakete installieren:

```
pip install flask
pip install flask-script
pip install WTForms
pip install mongoengine
pip install flask_mongoengine
pip install flask-Assets
```

Installation von MongoDB, Gunicorn, Supervisor und sonstiger Python-Pakete

- MongoDB installieren:

```
sudo apt-get install mongodb-server
sudo service mongod start
```

- Den Webserver Gunicorn installieren:

```
pip install gunicorn
```

- Supervisor installieren:

```
pip install supervisor
```

- Das Python-Paket für Zugriff auf Wetterdaten installieren:

```
pip install pyowm
```

Energie-Monitoring-System herunterladen und konfigurieren

- Software herunterladen:

```
cd /home
git clone https://github.com/artpetro/raspimeter.git
```

- Einen Zähler mit den Standardeinstellungen erstellen:

```
python /home/raspimeter/db/mongo_db_manager.py
```

- Die Konfigurationsdatei für Supervisor erstellen:

```
sudo nano /etc/supervisor/conf.d/raspimeter.conf
```

Die Datei enthält zwei Blöcke. In einem wird der Server gesteuert und in dem anderen - das Hauptprogramm des Monitoring-Systems:

```
[program:raspimeter_server]
command = gunicorn server:app -b 192.168.178.111:80
directory = /home/raspimeter/flask_server
user = pi
autorestart = true

[program:raspimeter_runner]
command = python /home/raspimeter/run/runner.py
user = pi
autorestart = true
```

Dabei ist 192.168.178.111 - die IP-Adresse von Raspberry Pi im lokalen Netzwerk.

- Die Konfigurationsdatei einlesen:

```
sudo supervisorctl reread
sudo supervisorctl update
```

- Den Server und das Programm starten:

```
sudo supervisorctl start raspimeter_server
sudo supervisorctl start raspimeter_runner
```

Nach diesen Schritten ist das Monitoring-System erfolgreich installiert. Die Weboberfläche ist unter der URL-Adresse <http://192.168.178.111> im lokalen Netzwerk erreichbar.

Anhang B

Glossar

ADC (Abkürzung für engl. *Analog-Digital-Converter*) „Ein Analog-Digital-Umsetzer ist ein elektronisches Gerät, Bauelement oder Teil eines Bauelements zur Umsetzung analoger Eingangssignale in einen digitalen Datenstrom, der dann weiterverarbeitet oder gespeichert werden kann.“¹

Arduino „ist eine aus Soft- und Hardware bestehende Physical-Computing-Plattform. Die Hardware besteht aus einem einfachen Board mit einem Mikrocontroller und analogen und digitalen Ein- und Ausgängen. Die Entwicklungsumgebung basiert auf Processing und soll auch technisch weniger Versierten den Zugang zur Programmierung und zu Mikrocontrollern erleichtern. Die Programmierung selbst erfolgt in C bzw. C++.“²

API (Abkürzung für engl. *Application Programming Interface*) Eine *API* ist eine dokumentierte Software-Schnittstelle, mit deren Hilfe ein Programm die Funktionen eines anderen Programms nutzen kann (gleiches gilt für die *API* eines Betriebssystems).

ARM (Abkürzung für engl. *Advanced RISC Machine*) „ist eine CPU-Architektur, die zu Beginn der achtziger Jahre von Acorn Computer entwickelt später dann gemeinsam mit Apple weiterentwickelt wurde. Es handelt sich um eine RISC-Architektur, die in vielen mobilen Endgeräten, in Handys, Handhelds und Notebooks aber auch als System-on-Chip (SoC) in den Modulen der Industrie-Computer, in Embedded Systemen, Spielkonsolen, Datenkommunikationsgeräten, Konsumergeräten, Kleinstgeräten und in der digitalen Signalverarbeitung eingesetzt wird. In Embedded-Systemen ist es der am meisten eingesetzte Mikroprozessor.“³

Einplatinenrechner (Engl. *SBC, Single-Board-Computer*) „ist ein Prozessorboard, das alle zum Betrieb des Systems erforderlichen Komponenten wie den Mikroprozessor, die Arbeitsspeicher, Caches, die Chipsätze usw. enthält. Weitergehende periphere

¹<https://de.wikipedia.org/wiki/Analog-Digital-Umsetzer>

²<https://de.wikipedia.org/wiki/Arduino>

³<http://www.itwissen.info/definition/lexikon/advanced-RISC-machine-ARM-ARM-Prozessor.html>

Funktionen für die Aus- und Eingabe, die Grafikdarstellung oder die Kommunikationssteuerung können über weitere Aufsteckmodule hinzugefügt werden.“⁴

Framework (objektorientiertes) Eine Menge kooperierender Klassen, welche die Elemente eines wiederverwendbaren Entwurfs für eine bestimmte Art von Software darstellen. Ein *Framework* bietet eine Architekturhilfe beim Aufteilen des Entwurfs in abstrakte Klassen und beim Definieren ihrer Zuständigkeiten und Interaktionen. Ein Entwickler passt das *Framework* für eine bestimmte Anwendung an, indem er Unterklassen der *Framework*-Klassen bildet und ihre Objekte zusammensetzt (Gamma u. a., 1996, S. 445).

HTML (Abkürzung für engl. *Hypertext Markup Language*) Standardisierte Seitenbeschreibungssprache für WWW-Seiten im Internet bzw. Intranet, welche von Charles F. Goldfarb entwickelt wurde und in der ISO-Norm 8879 definiert ist (auch \rightarrow XML). Sie definiert sowohl die Gestaltung, den Inhalt und die Grafik der Seite als auch die Hyperlinks zu eigenen oder fremden Seiten.

I2C „(Auch I²C, gesprochen „I quadrat C“) ist ein synchroner serieller Zweidraht-Bus (eine Daten- und eine Taktleitung), der für die Kommunikation zwischen ICs über kleine Distanzen geeignet ist. Entwickelt wurde er Anfang der 80er Jahre von Philips. I2C steht für IIC = Inter IC Bus. Aus Lizenzgründen heißt der I2C Bus bei manchen Herstellern auch TWI, two wire interface.“⁵

JSON „Die JavaScript Object Notation ist ein kompaktes Format zum Austausch von Daten, welches von Douglas Crockford definiert wurde. Die Basis bildet dabei eine Teilmenge der JavaScript Programmiersprache - konkret die JavaScript-Notation für Objektliterale - gemäß des ECMA-262 Standards (Dritte Edition, 1999). Trotzdem handelt es sich bei JSON um eine Vorgehensweise, die Daten sprachunabhängig auf einfache Weise strukturiert und die sich inzwischen für viele weitere Programmiersprachen wie Java, Programmiersprache C, C++ u.v.a. durchgesetzt hat.“⁶

MySQL „ist ein relationales Datenbankmanagementsystem (RDBMS), das auf allen gängigen PC-Betriebssystemen lauffähig ist. Es ist ein robustes und schnelles Mehrbenutzersystem für die Erstellung dynamischer Webseiten. MySQL ist das am weitesten im Web verbreitete Datenbankmanagementsystem (DBMS) mit dem Webseiten dynamisch geändert werden können.“⁷

NoSQL (Abkürzung für engl. *Not only SQL*) „bezeichnet Datenbanken, die einen nicht-relationalen Ansatz verfolgen und damit mit der langen Geschichte relationaler Datenbanken brechen. Diese Datenspeicher benötigen keine festgelegten Tabellenschemata und versuchen, Joins zu vermeiden, sie skalieren dabei horizontal.“⁸

⁴<http://www.itwissen.info/definition/lexikon/Einplatinen-Computer-SBC-single-board-computer.html>

⁵<http://www.mikrocontroller.net/articles/I2C>

⁶<http://www.itwissen.info/definition/lexikon/JSON-JavaScript-object-notation.html>

⁷<http://www.itwissen.info/definition/lexikon/MySQL.html>

⁸<https://de.wikipedia.org/wiki/NoSQL>

SPI (Abkürzung für engl. *Serial Peripheral Interface*) „ist ein von Motorola entwickeltes Bus-System mit einem „lockeren“ Standard für einen synchronen seriellen Datenbus (*Synchronous Serial Port*), mit dem digitale Schaltungen nach dem Master-Slave-Prinzip miteinander verbunden werden können.“⁹

UART (Abkürzung für engl. *Universal Asynchronous Receiver Transmitter*) „Eine UART-Schnittstelle dient zum Senden und Empfangen von Daten über eine Datenleitung und bildet den Standard der seriellen Schnittstellen an PCs und Mikrocontrollern.“¹⁰

XML (Abkürzung für engl. *eXtensible Markup Language*) *XML* ist eine Methode zur Repräsentation strukturierter Daten. *XML* ist (wie auch \rightarrow *HTML*) eine „vereinfachte“ Version der *Standard Generalized Markup Language* (SGML), die es Programmieren von Web-Seiten erleichtert SGML-Anwendungen zu schreiben, und dabei eigene Dokumententypen (DTD) festzulegen.

WSGI (Abkürzung für engl. *Web Server Gateway Interface*) „ist eine Schnittstellen-Spezifikation für die Programmiersprache Python, die eine Schnittstelle zwischen Webservern und Web Application Frameworks bzw. Web Application Servern festlegt, um die Portabilität von Webanwendungen auf unterschiedlichen Webservern zu fördern.“¹¹

⁹https://de.wikipedia.org/wiki/Serial_Peripheral_Interface

¹⁰https://de.wikipedia.org/wiki/Universal_Asynchronous_Receiver_Transmitter

¹¹https://de.wikipedia.org/wiki/Web_Server_Gateway_Interface

Literaturverzeichnis

- [Bass u. a. 2012] BASS, Len (Hrsg.) ; CLEMENTS, Paul (Hrsg.) ; KAZMAN, Rick (Hrsg.): *Software Architecture in Practice*. 3 edition. Addison-Wesley Professional, 2012. – 640 S. – ISBN 978-0321815736 58
- [Bradski 2008] BRADSKI, Adrian: *Learning OpenCV: Computer Vision with the OpenCV Library*. 1 edition. O'Reilly Associates, 2008. – 575 S. – ISBN 978-0596516130 38, 48
- [Bundesregierung 2016] BUNDESREGIERUNG: *Was bringt, was kostet die Energiewende*. 2016. – URL <https://www.bundesregierung.de/Content/DE/StatischeSeiten/Breg/Energiekonzept/0-Buehne/kosten-nutzen-energiewende.html>. – Zugriffsdatum: 10. Juli 2016 3
- [Chaffer und Swedberg 2012] CHAFFER, Jonathan ; SWEDBERG, Karl: *jQuery lernen und einsetzen: Bessere Webanwendungen mit einfachen JavaScript-Techniken entwickeln*. 1. Auflage. Dpunkt.verlag, 2012. – 398 S. – ISBN 978-3898647861 39
- [DeStatis 2008] DESTATIS: *Energieverbrauch der privaten Haushalte*. 2008. – URL https://www.destatis.de/DE/PresseService/Presse/Pressekonferenzen/2008/UGR/pressebroschuere_ugr.pdf?__blob=publicationFile. – Zugriffsdatum: 10. Juli 2016 4
- [DeStatis 2016a] DESTATIS: *Daten zur Energiepreisentwicklung*. 2016. – URL https://www.destatis.de/DE/Publikationen/Thematisch/Preise/Energiepreise/EnergiepreisentwicklungPDF_5619001.pdf?__blob=publicationFile. – Zugriffsdatum: 10. Juli 2016 3
- [DeStatis 2016b] DESTATIS: *Temperaturbereinigter Energieverbrauch der privaten Haushalte*. 2016. – URL https://www.destatis.de/DE/Publikationen/STATmagazin/Umwelt/2008_12/TabelleEnergieverbrauch.html. – Zugriffsdatum: 10. Juli 2016 5
- [Fastforward 2016] FASTFORWARD: *Offizielle Website*. 2016. – URL <http://fastforward.ag/>. – Zugriffsdatum: 10. Juli 2016 24
- [Fehrenbacher 2011] FEHRENBACHER, Katie: *5 reasons Google Power-Meter didnt take off*. 2011. – URL <https://gigaom.com/2011/06/26/5-reasons-google-powermeter-didnt-take-off/>. – Zugriffsdatum: 10. Juli 2016 5

- [Finanzen.net 2016] FINANZEN.NET: *Erdgaspreisentwicklung in Dollar (Chart)*. 2016. – URL <http://www.finanzen.net/rohstoffe/Erdgas-Preis-Natural-Gas/Chart>. – Zugriffsdatum: 10. Juli 2016 3
- [Flask 2016a] FLASK: *Flask-Erweiterungen*. 2016. – URL <http://flask.pocoo.org/extensions/>. – Zugriffsdatum: 10. Juli 2016 39
- [Flask 2016b] FLASK: *Form Validation with WTForms*. 2016. – URL <http://flask.pocoo.org/docs/0.11/patterns/wtforms/>. – Zugriffsdatum: 10. Juli 2016 65
- [Flask 2016c] FLASK: *Offizielle Website*. 2016. – URL <http://flask.pocoo.org/>. – Zugriffsdatum: 10. Juli 2016 38
- [Flask-Script 2016] FLASK-SCRIPT: *Offizielle Website*. 2016. – URL <https://flask-script.readthedocs.io/en/latest/>. – Zugriffsdatum: 10. Juli 2016 71
- [Fraunhofer 2011] FRAUNHOFER: *Presseinformation des Fraunhofer ISI*. 2011. – URL <http://www.isi.fraunhofer.de/isi-de/service/presseinfos/2011/pri11-13.php>. – Zugriffsdatum: 10. Juli 2016 5
- [Gamma u. a. 1996] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software*. 1. Aufl. München : Addison-Wesley, 1996. – xx + 479 S. – ISBN 3-89319-950-0 82
- [Gunicorn 2016] GUNICORN: *Offizielle Website*. 2016. – URL <http://gunicorn.org/>. – Zugriffsdatum: 10. Juli 2016 72
- [Highcharts 2016] HIGHCHARTS: *Offizielle Website*. 2016. – URL <http://www.highcharts.com/>. – Zugriffsdatum: 10. Juli 2016 39
- [Iberdrola 2016] IBERDROLA: *Frequently Asked Questions*. 2016. – URL <https://www.iberdroladistribucionelctrica.com/webibd/corporativa/iberdrola?IDPAG=ENSOCDISREDPRE>. – Zugriffsdatum: 10. Juli 2016 11
- [Jinja 2016] JINJA: *Offizielle Website*. 2016. – URL <http://jinja.pocoo.org/>. – Zugriffsdatum: 10. Juli 2016 71
- [jQuery 2016] JQUERY: *Offizielle Website*. 2016. – URL <https://jquery.com/>. – Zugriffsdatum: 10. Juli 2016 39
- [Juris 2005] JURIS: *Gesetz über die Elektrizitäts- und Gasversorgung*. 2005. – URL https://www.gesetze-im-internet.de/enwg_2005/. – Zugriffsdatum: 10. Juli 2016 11
- [Kickstarter 2016] KICKSTARTER: *SmartPi - Turn your Raspberry Pi into a SmartMeter*. 2016. – URL <https://www.kickstarter.com/projects/1240982104/smartpi-turn-your-raspberry-pi-into-a-smartmeter/description>. – Zugriffsdatum: 10. Juli 2016 12

- [Kompf 2016] KOMPf, Martin: *OpenCV Praxis: OCR für den Stromzähler*. 2016. – URL <http://www.kompf.de/cplus/emeocv.html>. – Zugriffsdatum: 10. Juli 2016 18
- [Lutz 2013] LUTZ, Mark: *Learning Python*. 5 edition. O'Reilly, 2013. – 1600 S. – ISBN 978-1449355692 37
- [Miner und Shook 2012] MINER, Donald ; SHOOK, Adam: *MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems*. 1 edition. O'Reilly Media, 2012. – 249 S. – ISBN 978-1449327170 55
- [MongoDB 2016] MONGODB: *Offizielle Website*. 2016. – URL <https://www.mongodb.com/de>. – Zugriffsdatum: 10. Juli 2016 38
- [MongoEngine 2016] MONGOENGINE: *Offizielle Website*. 2016. – URL <http://mongoengine.org/>. – Zugriffsdatum: 10. Juli 2016 54
- [nDenerserve 2016] NDENERSERVE: *Offizielle Website*. 2016. – URL <http://www.emanager.eu/de/home>. – Zugriffsdatum: 10. Juli 2016 11
- [NumPy 2016] NUMPY: *Offizielle Website*. 2016. – URL <http://www.numpy.org/>. – Zugriffsdatum: 10. Juli 2016 43
- [OpenCV 2016a] OPENCV: *Canny Edge Detector*. 2016. – URL http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html. – Zugriffsdatum: 10. Juli 2016 48
- [OpenCV 2016b] OPENCV: *Dokumentation*. 2016. – URL <http://opencv.org/documentation.html>. – Zugriffsdatum: 10. Juli 2016 38
- [OpenCV 2016c] OPENCV: *Image Processing*. 2016. – URL http://docs.opencv.org/2.4/doc/tutorials/imgproc/table_of_content_imgproc/table_of_content_imgproc.html. – Zugriffsdatum: 10. Juli 2016 48
- [OpenCV 2016d] OPENCV: *Machine Learning Overview*. 2016. – URL http://docs.opencv.org/3.1.0/dc/dd6/ml_intro.html#gsc.tab=0. – Zugriffsdatum: 10. Juli 2016 51
- [OpenCV 2016e] OPENCV: *Offizielle Website*. 2016. – URL <http://opencv.org/>. – Zugriffsdatum: 10. Juli 2016 38
- [OpenCV 2016f] OPENCV: *Template Matching*. 2016. – URL http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html. – Zugriffsdatum: 10. Juli 2016 51
- [OpenCV 2016g] OPENCV: *Understanding k-Nearest Neighbour*. 2016. – URL http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_ml/py_knn/py_knn_understanding/py_knn_understanding.html. – Zugriffsdatum: 10. Juli 2016 51

- [OpenEnergyMonitor 2016a] OPENENERGYMONITOR: *About*. 2016. – URL <https://openenergymonitor.org/emon/sustainable-energy>. – Zugriffsdatum: 10. Juli 2016 15
- [OpenEnergyMonitor 2016b] OPENENERGYMONITOR: *Community*. 2016. – URL <https://openenergymonitor.org/emon/community>. – Zugriffsdatum: 10. Juli 2016 16
- [OpenEnergyMonitor 2016c] OPENENERGYMONITOR: *User Guide*. 2016. – URL <https://guide.openenergymonitor.org/>. – Zugriffsdatum: 10. Juli 2016 16
- [OpenWeatherMap 2016] OPENWEATHERMAP: *Offizielle Website*. 2016. – URL <http://www.openweathermap.org/>. – Zugriffsdatum: 10. Juli 2016 41
- [PyOWM 2016] PYOWM: *Repository*. 2016. – URL <https://github.com/csparpa/pyowm>. – Zugriffsdatum: 10. Juli 2016 54
- [Python 2016] PYTHON: *Dokumentation*. 2016. – URL <https://www.python.org/doc/essays/blurb/>. – Zugriffsdatum: 10. Juli 2016 37
- [RaspberryPi 2016] RASPBERRYPI: *Raspberry Pi GPIO How-To*. 2016. – URL <http://raspberrypiguide.de/howtos/raspberry-pi-gpio-how-to/>. – Zugriffsdatum: 10. Juli 2016 46
- [Rodríguez u. a. 2014] RODRÍGUEZ, Miguel ; BERDUGO, Geovanni ; JABBA, Daladier ; CALLE, Maria ; JIMENO, Miguel: HD_MR: a new algorithm for number recognition in electrical meters. In: *Turkish Journal of Electrical Engineering and Computer Sciences* 22 (2014), Januar, Nr. 1, S. 87–96. – URL <http://journals.tubitak.gov.tr/elektrik/issues/elk-14-22-1/elk-22-1-8-1202-34.pdf>. – Zugriffsdatum: 10. Juli 2016. – ISSN 1300-0632 51
- [Schmolke u. a. 2013] SCHMOLKE, Siegfried (Hrsg.) ; DEITERMANN, Manfred (Hrsg.) ; RÜCKWART, Wolf-Dieter (Hrsg.): *Industrielles Rechnungswesen - IKR*. 45. Auflage. Winklers Verlag im Westermann Schulbuch, 2013. – 552 S. – ISBN 978-3804568525 5
- [Schutkin 2016] SCHUTKIN, Andey: *Erfassung eines Gaszählers mit Python und OpenCV*. 2016. – URL <https://habrahabr.ru/post/220869/>. – Zugriffsdatum: 10. Juli 2016 18
- [Smappee 2016] SMAPPEE: *Offizielle Website*. 2016. – URL <http://www.smappee.com/de/energiemonitor>. – Zugriffsdatum: 10. Juli 2016 14
- [Stromsparinitiative 2016] STROMSPARINITIATIVE: *Die Energiewende der Bundesregierung*. 2016. – URL <http://www.die-stromsparinitiative.de/stromkosten/energiewende-strompreise-und-eeg/index.html>. – Zugriffsdatum: 10. Juli 2016 4
- [Supervisor 2016] SUPERVISOR: *Offizielle Website*. 2016. – URL <http://supervisord.org/>. – Zugriffsdatum: 10. Juli 2016 56

- [SWO 2016] SWO: *Stadtwerke Osnabrück - Sparhilfen: smartWEB und smartCOCKPIT*. 2016. – URL <https://www.stadtwerke-osnabrueck.de/privatkunden/energie/strom/intelligenter-zaehler.html>. – Zugriffsdatum: 10. Juli 2016 10
- [Szeliski 2010] SZELISKI, Richard: *Computer Vision: Algorithms and Applications*. 2011 edition. Springer, 2010. – 812 S. – ISBN 978-1848829343 47, 48
- [Tesseract 2016] TESSERACT: *Repository*. 2016. – URL <https://github.com/tesseract-ocr>. – Zugriffsdatum: 10. Juli 2016 51
- [Trelle 2014] TRELLE, Tobias: *MongoDB: Der praktische Einstieg*. 1. Auflage. DPunkt.verlag, 2014. – 290 S. – ISBN 978-3864901539 38
- [Upton 2015] UPTON, Liz: *Senior Pi*. 2015. – URL <https://www.raspberrypi.org/blog/senior-pi/>. – Zugriffsdatum: 10. Juli 2016 29
- [Volkszaehler 2016a] VOLKSZAEHLER: *Eigenen Volkszaehler Installieren*. 2016. – URL <http://wiki.volkszaehler.org/doku.php/howto/getstarted>. – Zugriffsdatum: 10. Juli 2016 17
- [Volkszaehler 2016b] VOLKSZAEHLER: *Offizielle Website*. 2016. – URL <http://volkszaehler.org/>. – Zugriffsdatum: 10. Juli 2016 17
- [Walerowsky 2007] WALEROWSKY, Peter: *Python: Grundlagen und Praxis*. 1. Auflage. Addison-Wesley, 2007. – 336 S. – ISBN 978-3827325174 43
- [Wikipedia 2016a] WIKIPEDIA: *Google PowerMeter*. 2016. – URL https://en.wikipedia.org/wiki/Google_PowerMeter. – Zugriffsdatum: 10. Juli 2016 5
- [Wikipedia 2016b] WIKIPEDIA: *Intelligenter Zähler*. 2016. – URL https://de.wikipedia.org/wiki/Intelligenter_Zähler. – Zugriffsdatum: 10. Juli 2016 9
- [Wikipedia 2016c] WIKIPEDIA: *S0-Schnittstelle*. 2016. – URL <https://de.wikipedia.org/wiki/S0-Schnittstelle>. – Zugriffsdatum: 10. Juli 2016 21

