UNIVERSITY OF TARTU
Faculty of Computer Science
Conversion Master in IT curriculum

Peeter Liik

Live data prototype of a mechanical water-meter digitiser

Master Thesis (15 EAP)

Supervisor: Ahmed Aboumalwa (MSc)
Co-supervisor: Heili Orav (PhD)

Tartu 2022

Live data prototype of a mechanical water-meter digitiser

Abstract

Remote reading using smart metering is the modern and efficient way of gathering consumption data from the users of water, electricity, gas, and other utilities. Smart metering has eliminated the need for the user to report the readings manually to the utility company. On the other hand, it has assured the utility company of a consistent and reliable data feed.

Although many households in Estonia enjoy the comfort of smart metering for different utilities, smart metering water consumption seems to be lagging behind the others. Most Estonian households, especially the apartments, still report water readings manually.

This thesis aims to build a live data prototype of a digitiser that would add the main benefits of a smart meter, remote reading, and consumption history to a regular mechanical analogue water meter. Thus, serving as a viable low-cost intermediate substitute to an actual smart meter.

Building a live data prototype proved that making a low-cost substitute for a smart meter using optical character recognition tool to turn analogue readings from a picture into machine-readable data is possible, and this solution can be turned into a product.

Keywords: Smart metering, Remote reading, OCR, Digitisation, Node.JS, Raspberry Pi
CERCS: P170 (Computer science, numerical analysis, systems, control)

Mehhaanilise veearvesti digiteerija reaalaja andmete prototüüp

Lühikokkuvõte

Tarkade arvestite kasutamine tarbeandmete kauglugemise eesmärgil on moodne ja efektiivne lahendus erinevate tarbimisandmete nagu vesi, elekter, gaas ja muu kogumiseks, töötlemiseks ja edastamiseks. Tänu tarkadele arvestitele on kadunud vajadus tarbimisnäitude käsitsi teatamise järele. Elektri, vee ja gaasiettevõtted saavad aga tänu kauglugemisele oluliselt täpsemad ja usaldusväärsemad andmed.

Kuigi mitmed majapidamised Eestis naudivad täna kauglugemise hüvesid mitmete erinevate tarbeliikide puhul, on vee tarbeandmete kauglugemine võrreldes mõne teise tarbeliigiga olulises mahajäämuses. Valdav enamik Eesti majapidamisi, eriti korterid, edastavad näite endiselt käsitsi.

Antud töö eesmärk on ehitada reaalajas andmeid edastava digiteerija prototüüp, mis lisaks tavapärasele mehhaanilisele analoognäidikuga veearvestile peamised targa arvesti kasutunnused, nagu kauglugemine ja tarbeandmete ajaloo talletamise ning kuvamise võimaluse. Eesmärk on luua seade, mis oleks madalahinnaline täisfunktsionaalne asendus targale veearvestile.

Digiteerija ehitamise protsess ja lõpptulemus näitasid, et madalahinnalise targa arvesti asenduslahenduse ehitamine on võimalik, kasutades optilise märgituvastuse tööriistu fotolt näidu tuvastamiseks ning selle abil näidud masinloetavasse vormi teisendades. Toimiva prototüübi ehitamine näitas, et lahendust on võimalik tootestada.

Võtmesõnad: tark arvesti, kaugloetav arvesti, optiline märgituvastus, digiteerimine, Node.JS, Raspberry Pi

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Introduction

Metering of different kinds of residential or industrial cost items like electricity, gas, or water in Estonia has, in general, introduced the technology of remote reading.

Estonian Grid Code states (Riigi Teataja, 2019) that starting from the 1st of January 2017, every metering point with the circuit breakers value under 63A has to be able to be measured remotely. Meaning that today almost all of Estonia's electricity metering points are measured remotely using smart meters.

Gas metering is required by law to be remotely readable starting from 2021 (Pott, 2019). However, gas meters inside apartments in apartment buildings are not obliged to have remotely readable meters.

Measuring water consumption in an average household is lagging in the sense of digital technologies and remote reading. Tallinna Vesi, a company responsible for ensuring the service of water for the inhabitants of Tallinn and surrounding areas, for a total of 470 000 customers, has stated that the installation of smart water meters around their area of responsibility started in 2021. Their aim is to replace all the traditional mechanical water meters with smart ones by the year 2026 (Tallinna Vesi, 2022). Pärnu Vesi (around 44 000 customers in Pärnu and surrounding areas) started to install remote meters in 2018 and is planning to finish the job and thus reach all its customers by 2028 (Pärnu Vesi, 2021). There are about 50 companies (Eesti Vee-ettevõtete Liit, 2022) providing water service in Estonia. All of them are different in size, areas of service, the number of customers to be served, and variable strategic plans for technological development. Many have no publicly stated strategy to invest in smart meters.

It is unclear when will a country like Estonia reach the same sort of remote metering coverage for water metering as it is currently present in the electrical grid. However, it could be argued that for at least the next ten years, a lot of households are still forced to have the traditional mechanical water meters and use the manual readings reporting practice.

Remote reading is generally favourable for the users and suppliers as it dramatically reduces the overhead and manual work involved in reporting and handling the metering data. With mechanical water meters, the value of the reading has to be visibly checked from the water meter gauge every month by the customer and then manually sent to a predefined email address or reported on a service provider's homepage.

It has many disadvantages. For one, a customer must remember to check the reading periodically, preferably during the same time of the month, to ensure consistency in reporting statistics. Additionally, manual reporting is prone to errors, like with every manual work that requires human input. This means a lot of unreliable data that is also not in real-time for the service provider.

Remote reading, on the contrary, means that there is no need for customer input and that the measurement data is reported automatically to the service provider.

The additional benefits of remote reading by using smart metering technologies are also more in-depth consumption monitoring in real-time and the ability for the consumer to adjust consumption according to the real-time price adjustments (Cook, B. et al., 2013).

The project described in the thesis aims to build an experimental prototype of a digitiser that will transform mechanical water meter readings from the analogue gauge into a machine-readable form by using optical character recognition software. Additionally, the prototype will handle the data in real-time by producing a water consumption dashboard and a readings reporting scheduler that will automatically forward consumption data to a predefined e-mail address with a predefined interval.

The device, together with the software, could serve as a transition area supplement for a mechanical water meter to imitate the data reading functions of a smart water meter. Thus giving the consumer all the benefits of a smart meter and providing the service provider with reliable measuring data.

The focus is strictly on making the consumer's life more convenient by not forcing the customer to report readings to the service provider manually. And also to offer the customer

opportunity to follow its consumption data in real-time and thus make more environmentally friendly and financially adequate decisions.

1. Background - Water metering technologies

The basis for remote reading is the digitisation of the reading data. Making the consumption data machine-readable makes it universally available for any sort of reading automation. Next is a brief overview of the two water metering technologies - mechanical and smart metering and the main difference in data extraction.

1.1 Mechanical water meters

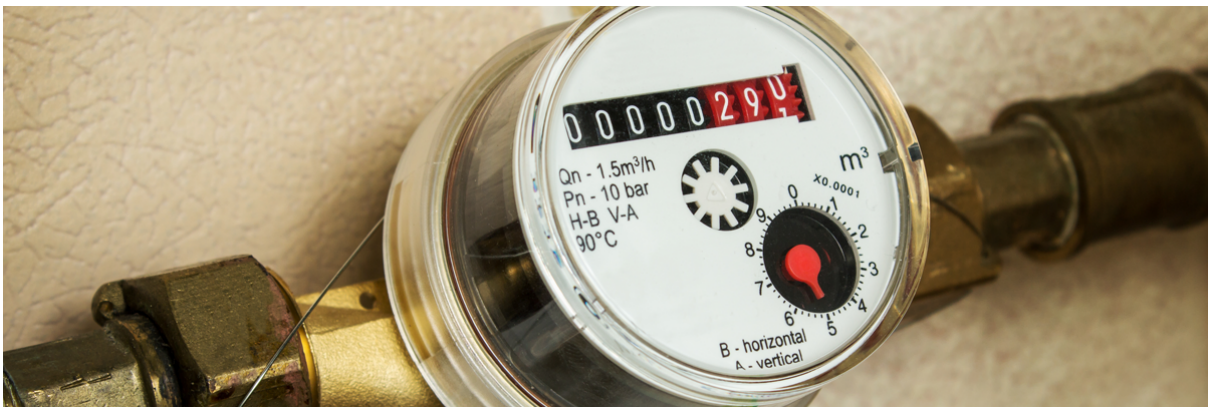


Figure 1. Mechanical water meter (The NoBroker Times, 2022)

There are two significant methods of flow measurement in use for mechanical water meters - displacement and velocity. Most widely used displacement designs include oscillating pistons and nutating disk meters. Velocity-based designs are mostly jet meters and turbine meters (Monica, Crainic, 2012).

A positive displacement water meter “measures the volume flow rate of a continuous flow stream by momentarily entrapping a segment of the fluid into a chamber of known volume and releasing that fluid back into the flow stream on the discharge side of the meter. By monitoring the number of entrapments for a known period of time or number of entrapments per unit time, the total volume of flow or the flow rate of the stream can be ascertained.” (Monica, Crainic, 2012).

“A velocity-type meter measures the velocity of flow through a meter of a known internal capacity. The speed of the flow can then be converted into the volume of flow for usage”(Monica, Crainic, 2012).

The measured amount for both types is displayed on an analogue counter gauge. The measurement is not saved anywhere, and it only shows a local real-time counter value. Because of the closed communicative nature of these kinds of water meters, all the readings have to be recorded by a visual inspection done manually by a human being.

1.2 Smart water meters

Figure 2. Kamstrup Multical 21 - smart water meter (Kamstrup, 2022)

Estonian water companies widely use a smart water meter produced by the Danish company Kamstrup (Eesti Vee-ettevõtete Liit, 2022). These devices have no moving parts, and they measure the amount of flow with digital sensors. They ship together with data transmitting modules that transmit the measurement data in real-time wirelessly on 868-870 Mhz (Kamstrup, 2021). This also means that the data can be saved directly to the database and used to build consumption history graphs. The data can be accessed in real-time, both by the service provider and the consumer.

2. Requirements for the prototype

The following chapter will list the predefined requirements for the prototype. These are set while taking into account the limitations that a prototype of a product will inevitably have. However, in order to test the concept, the requirements are set high in order to get all the functional benefits of an actual smart meter.

2.1 Type of prototype

The aim is to build a live data prototype. Live data prototypes are best used for experimental products to prove that the concept works. Live data means that there is access to actual data, and it is sent in real life. A live data prototype is not a ready product but rather a proof of concept with a degree of qualitative and quantitative scalability (Cagan, 2014). It could be argued that a working live data prototype is the closest step to an actual commercially sold product. Although the amount of work and development needed to get to a real product can still be significant.

2.2 Input data

Readings have to be transformed into machine-readable form in order for the data to be usable. Mechanical water meters display an analogue counter. The reading from that counter should be transformed into a digital machine-readable number value that can be saved to a database.
Data has to be recorded from the analogue counter at least once an hour to ensure fault tolerance and enough usable data for the consumption dashboard.

2.3 Presentation

Data should be presented on a real-time dashboard that includes at least the following attributes:

1) Latest reading
2) Consumption in the current month
3) A monthly consumption graph

The dashboard can have more than one display for each to give a more in-depth overview of the consumption history.

2.4 Automation

The application should include a possibility to send out the reading to a freely chosen e-mail address. There should also be a possibility to schedule a periodical sending to a chosen e-mail address.

2.5 Hardware

The hardware used for the prototype should be commercially sold to ensure its availability. The equipment should be compact and should be able to produce a constant, uninterrupted live data feed for the live data prototype. The equipment doesn't need to be turned directly into a ready product. Still, it should at least give a fair estimate of what sort of physical measurements, communicative properties, and power consumption needs have to be taken into account when turning the prototype into a product.

2.6 Budget

The budget for the whole project should not exceed 60€. It is essential because the cost of the digitiser together with the software should remain significantly lower than an actual water consumption smart meter to be considered a viable alternative.
The costs of smart meters can vary significantly. For example, in Washington DC, the USA, a local water company DC Water has reported that adopting smart water meters costs 180$ per household (Snow, 2017). The actual cost of a smart meter can be even higher than that. For example, a company called Optimatic (Küttemaailm OÜ), which has been creating remote reading systems in Estonia for more than fourteen years, states in its price list that the prices for smart meters start from 131€ (without VAT) but can reach up to 1175€. These prices do not include installation costs (Optimatic, 2022).

3. Technology

The following chapter will introduce the basic technological concepts, hardware and software used in the prototype.

3.1 Optical Character Recognition

The current project uses Optical Character Recognition (OCR) to fill the requirement of turning input data from analogue counter to machine-readable digital info.
OCR is a process that converts images of handwritten or typed text into machine-readable form. The process of OCR involves two main parts:

a) hardware that records a digital image of the physical text. Either by taking a photo of it, scanning it, or using any other means of digitisation on a physical text;
b) software that processes the image singles out the letters and numbers, and turns them into machine-readable values. The software part of the process typically involves trainable artificial intelligence (AI) in order to detect characters with more precision and to take into account the physical properties of the text visible on the image (IBM Cloud Education, 2022).

3.2 Hardware

3.2.1 Raspberry Pi Zero 2 W and ZeroCam NightVision camera module

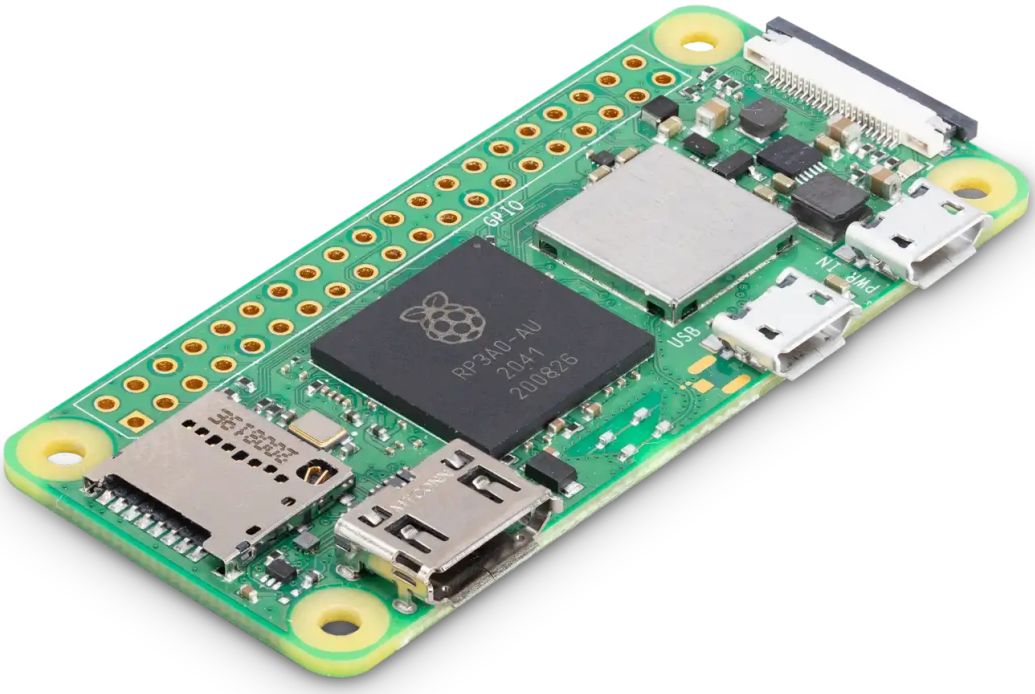


Figure 3. Raspberry Pi Zero 2 W (Raspberry Pi, 2022)

In this project, Raspberry Pi Zero 2 W[1] together with the ZeroCam NightVision[2] camera module is used in order to record the physical value from the analogue counter, process the image, do character recognition, and place the value into a database. A database of recorded values will be stored on the Raspberry Pi Zero internal storage and the computer will also act as a server for the backend service.

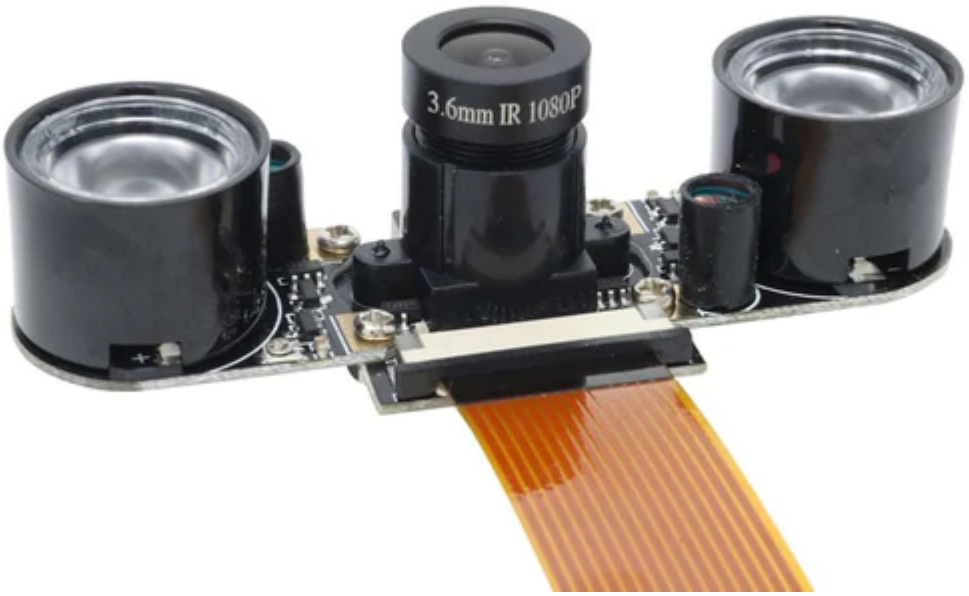


Figure 4. ZeroCam NightVision for Raspberry Pi (The PiHut, 2022)

ZeroCam NightVision is an infrared camera and it is chosen because the camera needs to operate in conditions where there is no visible light.

The particular model of Raspberry Pi is used because of its affordable price (13€)[3], compact size, and wireless internet capabilities.

Raspberry Pi is a UK-based company whose aim is to drive down the costs of general-purpose computing. They produce small computers called Raspberry Pi-s that are generally very affordable (starting from 3€) and run on Linux OS (RaspBerry Pi, n.d.). Raspberry Pi computers are especially popular among hobbyists and for educational purposes when building innovative projects.

[1] https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/
[2] https://thepihut.com/products/zerocam-nightvision-for-pizero-raspberry-pi-3
[3] https://thepihut.com/collections/raspberry-pi/products/raspberry-pi-zero-2

3.3 Software

3.3.1 Tesseract

In this project, an OCR software called Tesseract will be used. This particular software is chosen because of its open-source license, its popular use, and its active developer community.

The original developer of Tesseract was Hewlett-Packard (HP) Laboratories Bristol. It was open sourced by HP. From 2006 until 2018 it was developed by Google (Tesseract, n.d.).

3.3.2 Microservices

The code for the project is written while taking into account the possibilities of potential productizing and scaling. For that, the whole software part of the solution is written as independent microservices. This will ensure that the code will be easily manageable also in the future with a bigger team as services can also be developed totally independently. The microservices approach also allows using different coding languages for different services if necessary (LogicMonitor, n.d.).

“ In software development, microservices are an architectural style that structure applications as a collection of loosely connected services, making it easier for developers to build and scale apps. The microservices architectural approach differs from the conventional monolithic style, which treats software development as a single unit.” (LogicMonitor, n.d.).

3.3.3 Language

3.3.3.1 Python

Python is used to write the code for Raspberry Pi which comes built-in with the Operating System (Raspberry Pi OS, n.d.). Thus, the popular coding language can be considered a native for this type of computer. Also, the camera module controls ship natively with a python interface called PiCamera[4].

[4] https://picamera.readthedocs.io/en/release-1.13/

These aspects make Python a favourable and logical choice for writing code for the Raspberry Pi hardware.

Python is an object-oriented programming language. It uses a clear and rather simple syntax which makes it a great choice for the first language to learn. It comes with a large standard library that covers most of the common functionalities needed for application development - web server, file management, and text searching with regular expressions (Python, n.d.) . It is not only popular as a beginner's programming language but actually widely used in the professional industry. As of May 2022, Python is, according to the PYPL Popularity of Programming Languages index, the most popular programming language in the world (PYPL, 2022).

“Python is used by Intel, IBM, NASA, Pixar, Netflix, Facebook, JP Morgan Chase, Spotify, and a number of other massive companies. It’s one of the four main languages at Google, while Google’s YouTube is largely written in Python” (Python Developer, 2022).

3.3.3.2 JavaScript

JavaScript is chosen as a common language for backend and frontend services. JavaScripts versatility as a suitable language for full-stack development is one of the main reasons why it's used for this project.
Additionally, for accessibility reasons, the dashboard interface should be a web application, and JavaScript has some of the industry-standard runtimes and libraries for that purpose. It’s also the only language that is native to web browsers (Hack Reactor, 2021).

As of May 2022, JavaScript is, according to the PYPL Popularity of Programming Languages index, the third most popular programming language in the world (PYPL, 2022).

3.3.3.3 Node.js

A JavaScript runtime called Node.js will be used as the basis of JavaScript programming for the project. Node.js is built on Chrome’s V8 JavaScript engine. It’s an asynchronous event-driven runtime that is designed to build scalable network applications (Node.js, n.d.). For that reason, Node.js is a great option to choose when building the current live data

prototype. Additionally, taking into account the potential future productizing stage, the code used in the prototype would be scalable and reusable when written using Node.js.

3.3.4 Significant libraries

The project also uses different popular libraries. The following are the most essential for the project's cause.

3.3.4.1 React js

React.js library will be used to build the frontend. React.js is component-based and uses declarative methods of coding (React, n.d.). This makes it a great choice when there is a need to build a simple but effective and informative dashboard that uses repetitive design elements (i.e. information cards or boxes), that can be derived from a single reusable interactive component.

3.3.4.2 MongoDB

To store the reading values, a document database called MongoDB is used. Document databases are more flexible than SQL databases. They allow more variations in the document structure and they allow storing documents that are partially completed (MongoDb, n.d.). In the terms of the current project, this is valuable for testing purposes as it also allows potential faulty readings for further analysis.

3.3.4.3 Express

In order to create routing between the microservices, a popular web framework called Express[5] is used. Express lets to build a full REST API for the backend service that forms the basis of database query architecture.

3.3.4.4 Nodemailer

One distinctive requirement of the application is the ability to send out emails. Nodemailer[6] is dedicated to that exact purpose - it lets you send out emails from a Node.js application.

[5] https://www.npmjs.com/package/express
[6] https://www.npmjs.com/package/nodemailer

3.3.5 Design

3.3.5.1 Figma

To design the water meter dashboard, a web-based design application called Figma is used. It's an easy to use application that is specifically meant for designing desktop and mobile web interfaces[7].

3.3.6 Best coding practices

This project uses best coding practices collected and written down by a company called Pipedrive, specifically a team called Rigas Tribe, who develops a bulk mailing application named Campaigns for the Pipedrive company.
These specific coding practices are used because Pipedrive uses similar technology for its application that will be used for the prototype. And also because the author of this thesis has access to those practices by working in Rigas Tribe, Pipedrive. Rigas Tribe's best coding practices are not publicly available documents.

3.3.7 Security and live access to the database

Microservices and the database of this prototype will be built in a way that these will be scalable and available to be uploaded to a live hosting site for the application to work as a publicly available web application.
However, in order to assure the security of data and the services, this would need a thorough and secure authentication module for user management. As this is not part of the current prototype requirements, user management will not be included in the scope of this live data prototype.
For security reasons, the database or any of the services of this prototype will not be hosted on any publicly available hosting site, but the prototypes work will be tested and demonstrated locally on two machines - Raspberry Pi (readings recognition and database) and one PC (frontend and backend of the dashboard).

[7] https://www.figma.com/

3.3.7.1 ngrok

To ensure live access to the database, a free reverse proxy service called ngrok[8] will be used to produce a secure direct line connection between the database, hosted on Raspberry Pi, and the Server, that is hosting backend and frontend services.

4. Results

The following chapter describes the end result - on how to use the prototype and also how the prototype works on a conceptual level and on a code level.

[8] https://ngrok.com/

4.1 Process flow diagram

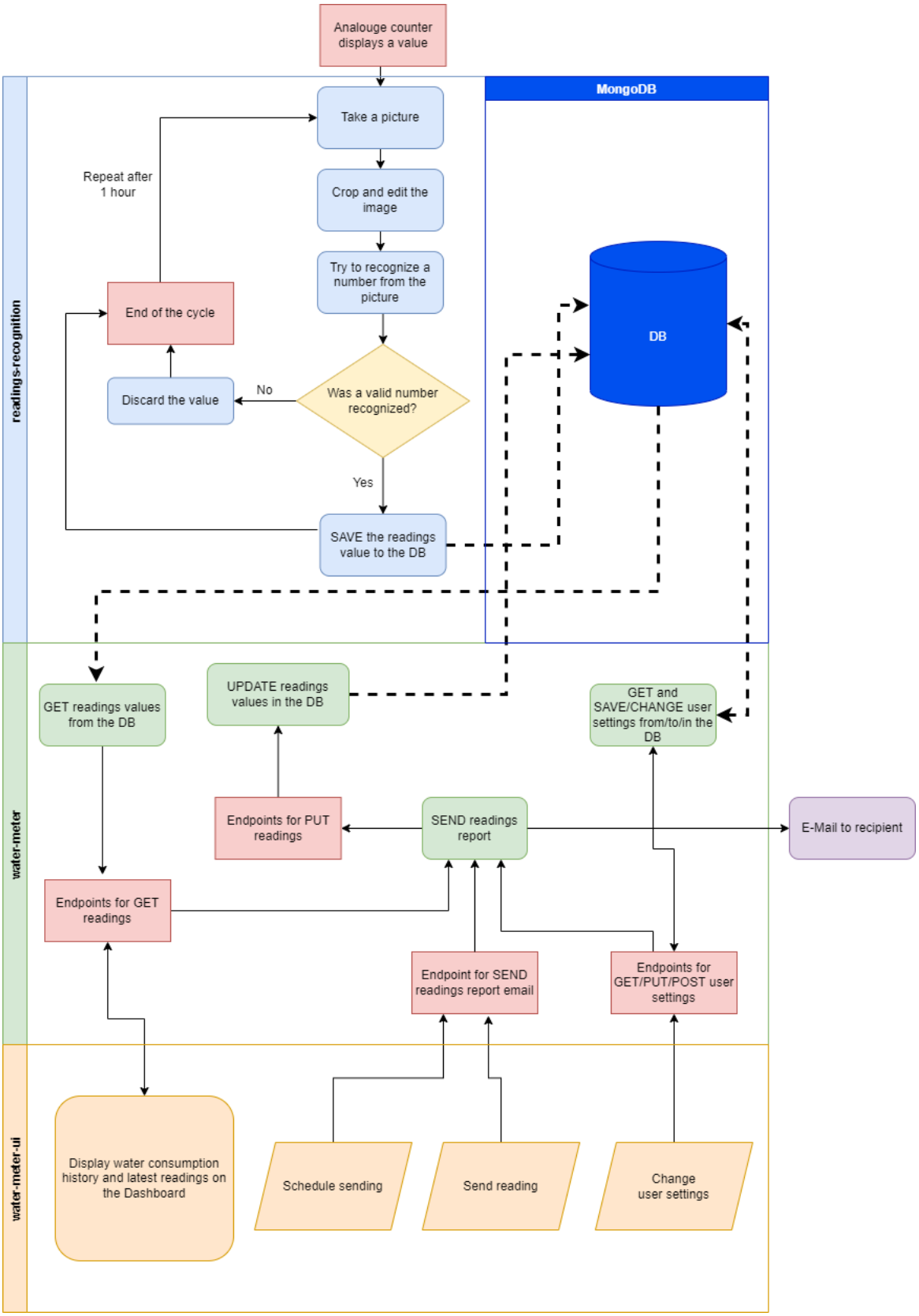


Figure 5. Process flow diagram of the water meter digitiser

Figure 5 shows the full flow of the process of digitisation. It starts from a mechanical water meter analogue counter and ends with the reading values being visible on the dashboard display.

The Flow diagram is divided into two main parts by the hosting machine:

1. **Raspberry Pi**
    a. **readings-recognition** is responsible for transforming the counter values into the machine-readable form and saving them to the database.
    b. **MongoDB** holds the database for the prototype.
2. **Server**
    a. **water-meter** is responsible for communicating with the DB in order to get/change/post values for the readings and the user settings.
    b. **water-meter-ui** is responsible for displaying the user interface and also communicating user inputs to the water-meter service.

4.2 Using the prototype

4.2.1 Hardware Setup

Raspberry Pi, together with the camera module, is installed on a tripod on the floor of the bathroom, the camera directed towards the analogue water counter of the mechanical meter. The camera is horizontally on the same level as the face of the counter. The distance between the camera and the counter is 6.4cm (Figure 6).

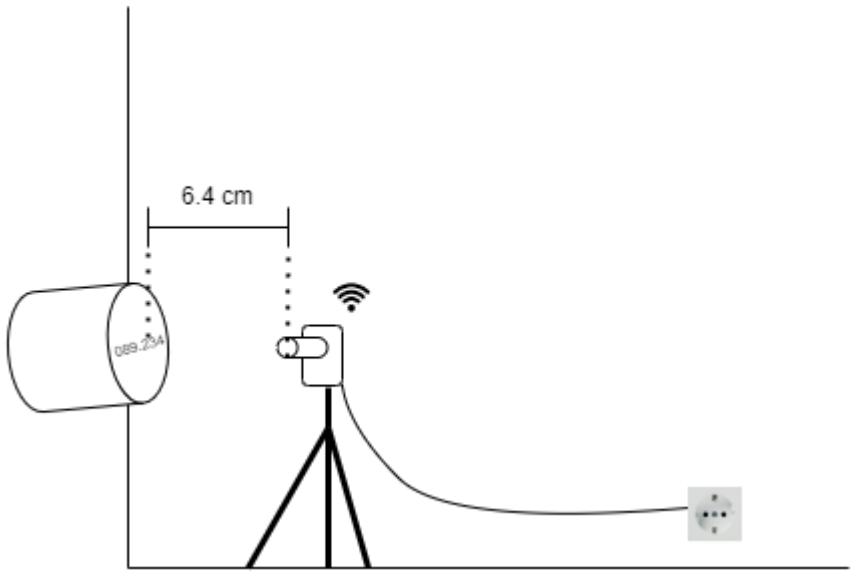


Figure 6. Hardware setup of the water meter prototype

This proved to be an optimal position to achieve a focused snapshot of the counter reading (Figure 7). For this prototype, a ZeroCam camera module with a focal length of 3.6mm was used and it takes pictures or video with a resolution of 1920x1080px[9]. If there is a need to take pictures from a closer distance, then a FishEye version of the same camera could be used that has a focal length of 1.4mm[10].

9 https://thepihut.com/collections/zerocam/products/zerocam-nightvision-for-pizero-raspberry-pi-3

10 https://thepihut.com/collections/zerocam/products/zerocam-fisheye-nightvision-for-pizero-raspberry-pi-3

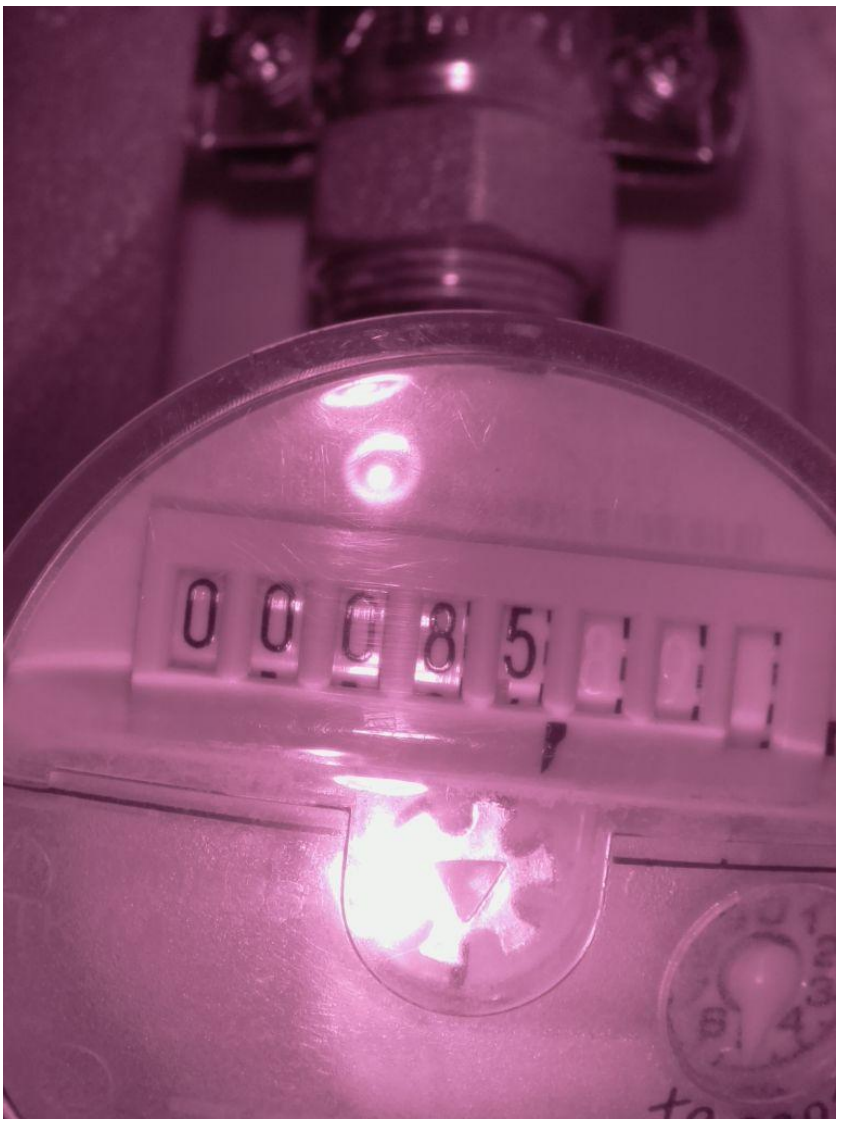


Figure 7. Photo of the counter value taken by the Raspberry Pi Camera. Near zero light conditions.

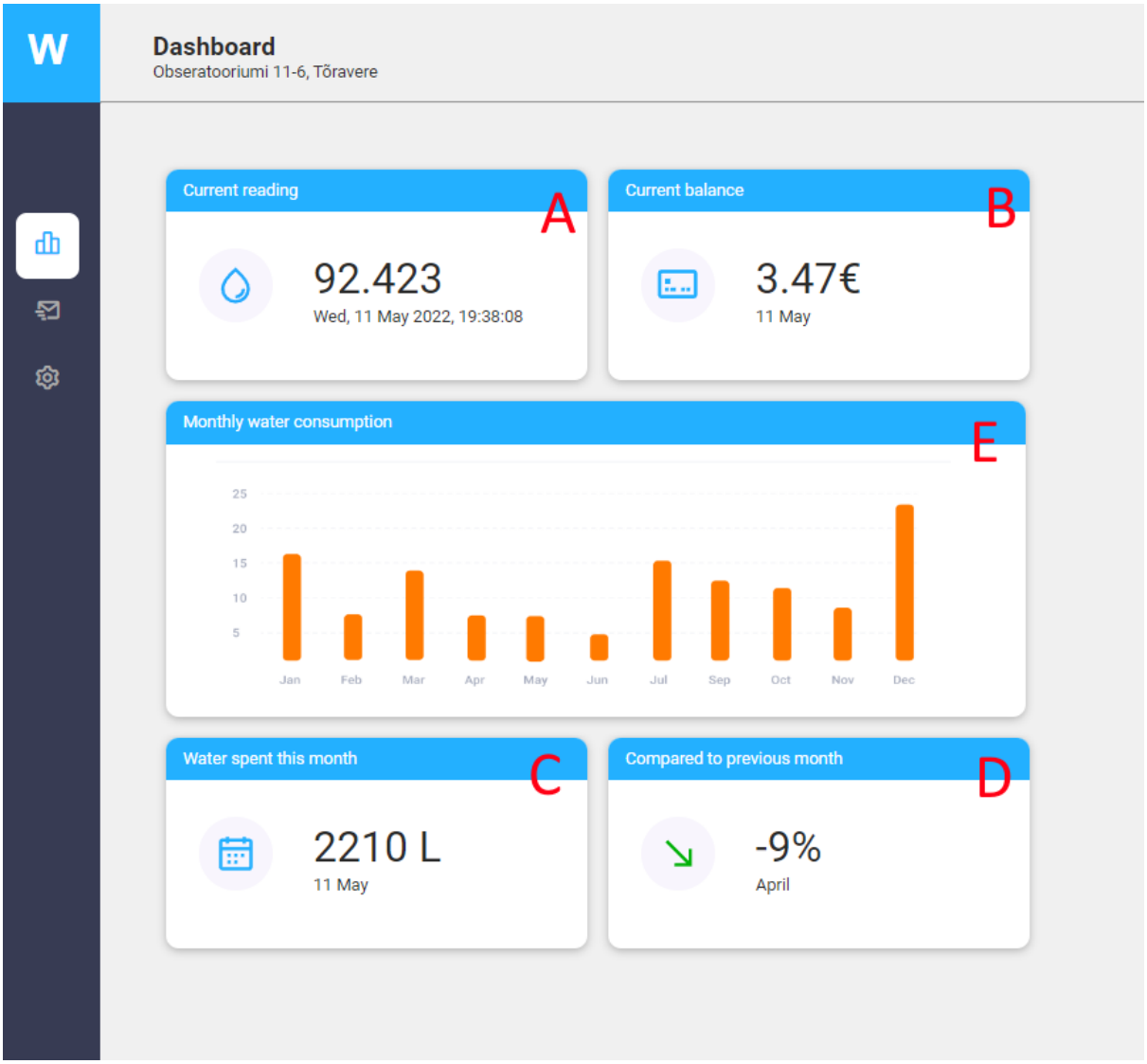
Raspberry Pi is powered by a regular power outlet with an adapter of 5.1V and 3A. It is connected to the internet using WiFi.

4.2.2 Using the UI

The UI has three different menu options. These can be chosen from the lefthand panel.

4.2.2.1 Dashboard

The opening screen of the UI takes the user to the Dashboard. Five different displays can be observed there.



(A) Current reading - a display of the latest valid reading with the time of recording. The camera takes a picture of the reading every hour. If the value is correctly recognised from the picture, the value is valid and is saved to the database. If the value is not recognised from the picture, the value is invalid and will not be saved to the database.

(B) Current balance - displays the running costs of the current month in euros. Takes the amount of water spent in the current month and calculates the price of consumption using the price of water per cubical meter.

(C) Water spent this month - the amount of water used during the current month. Displays the amount of water in litres and also the calculation date. The latest valid reading is used for the calculation.

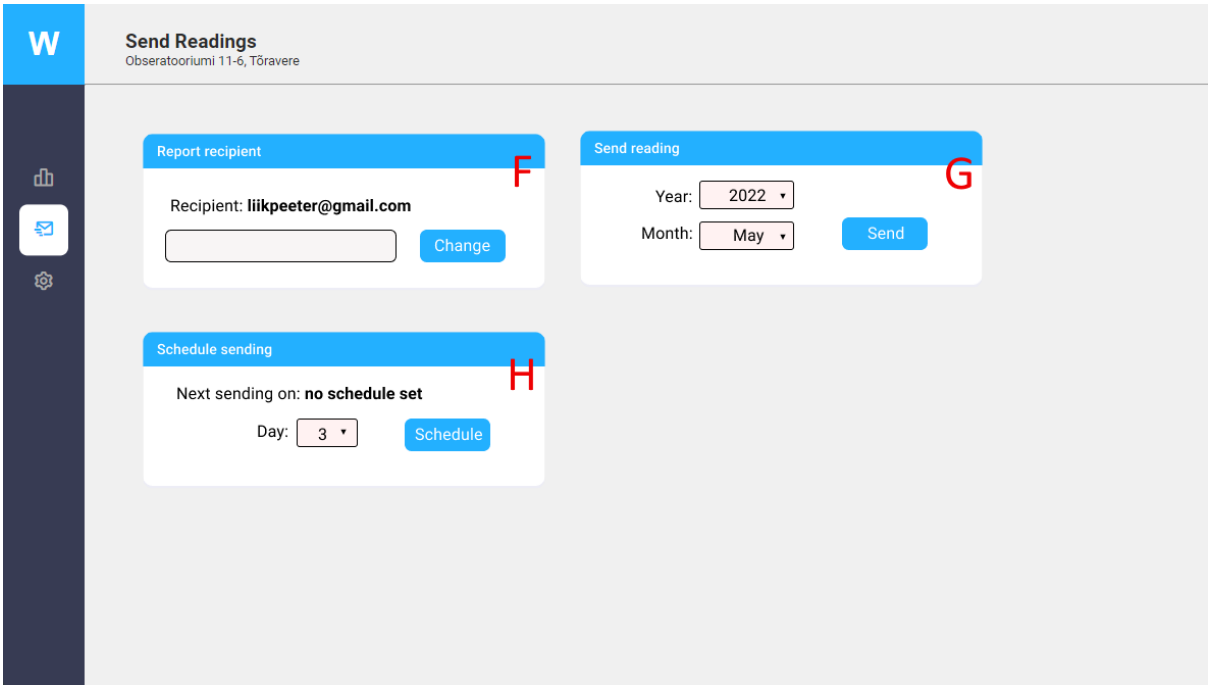
(D) Compared to the previous month - displays the difference in monthly water consumption compared to the previous month in percentages. It compares the amount of

water spent during the month to date with the amount of water spent during the last month until the same date.

(E) Monthly water consumption - displays a graph that depicts monthly water consumption for the last twelve months.

4.2.2.2 Send readings

The user has the opportunity to report the readings on these pages. Reporting can be done by sending the readings to a chosen email address. The page has three action cards.



(F) Report recipient. To send a readings report, a recipient email address has to be chosen. The *recipient address* field shows the current chosen recipient email address. To change that, a new address has to be written on the input field, and *Change* button has to be clicked. This will display the new recipient address, and it will be used for all the reading reportings.

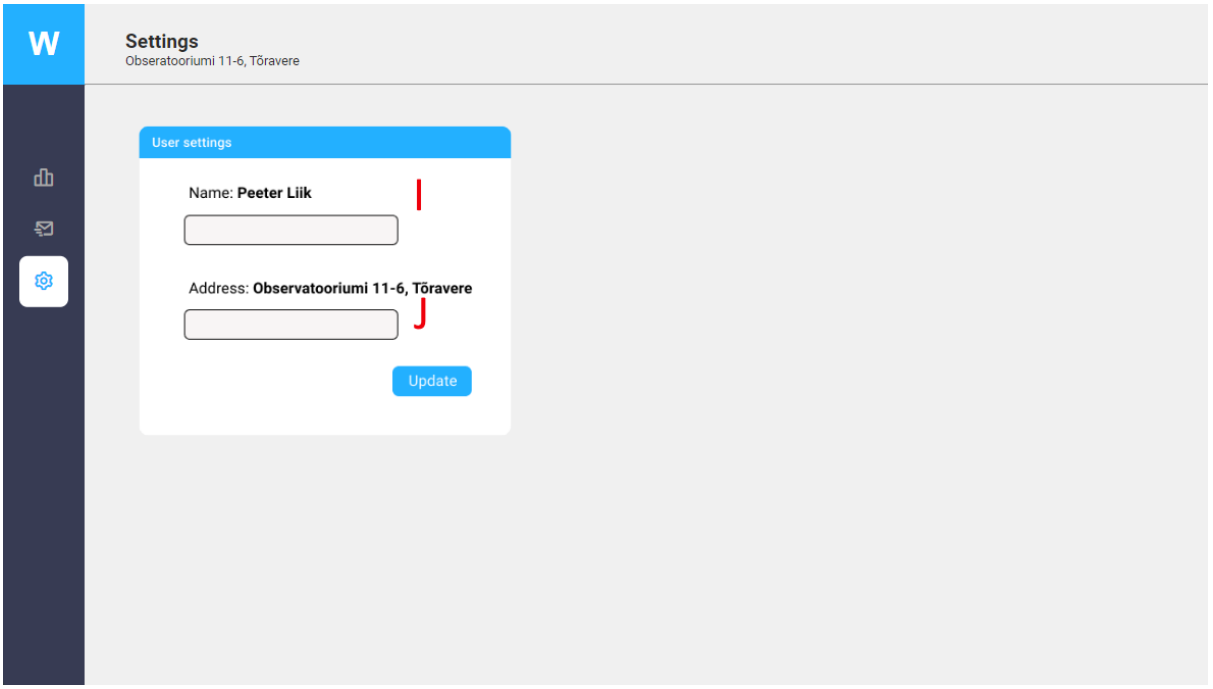
(G) Send reading. To send a reading, the user has to choose a month and a year of the water meter reading. By default, the current month is chosen, but the user can also send any of the previous month's readings. Pushing the *send* button will send the reading to the email address saved on the previous card.

(H) Schedule sending. In order to fully automate reading reporting, the user can schedule sending reading reports. The user can pick a day of the month when the reading will be sent

every month. When saved, the schedule info will be displayed. This schedule will run indefinitely if not cancelled from the same page. Schedule send will use the same recipient address that was input in the previous section.

4.2.2.3 Settings

On the settings page, the user can change the information that goes along with the reading reporting.



(I) Name of the sender. Name of the person, who is reporting the water meter readings.
(J) Home address. Address of the house/apart/ent where the readings are recorded from.

4.2.3 Description of microservices

The prototype code is divided into three separate and independently working microservices.

4.2.3.1 readings-recognition

Github link: https://github.com/peeterliik/reading-recognition

The service is working inside the Raspberry Pi and is written in Python. Raspberry Pi is running the Raspberry Pi OS kernel version 5.15[11]. The service implements an hourly cron-job that implements the following set of events:

[11] https://www.raspberrypi.com/software/operating-systems/#raspberry-pi-os-32-bit

1. **Takes a picture.** The picture is taken using the OS built-in package called picamera[12] that provides a Python interface to the camera module. A resolution of 1024x768px is used. The pictures are not saved indefinitely. With every new cron interval, the picture file is overwritten to avoid filling up the local storage space.
2. **Crops and edits the picture.** The background is removed from the picture as much as possible, the picture is cropped in a way that only the value remains. All colours are removed from the picture and the contrast is turned up. The editing and cropping purpose is to make the value stand out from the picture as much as possible. Scikit-image[13] is used for cropping and editing the picture.
3. **Performs optical character recognition**. The picture is then fed into the character recognition tool called pytesseract[14] that performs character detection. Image to string method is perfomed on the picture using settings that specify that the picture is in black and white colors and is only depicting numbers.
4. **Saves the value to the database.** Local MongoDB is used for storing the readings on the Raspberry Pi. A document with three fields is inserted into the database. Fourth one (unique ID) is added by MongoDb:

   {

   id: [unique ID of the document]

   value: [value detected from the picuture] (of type string)

   date: [date and time of when the value was inserted to the database] (of type Date)

   sent: [boolean value if the value has been sent out by email] (of type boolean, default is False)

   }

4.2.3.2 water-meter

Github link: https://github.com/peeterliik/water-meter

The service runs on a PC and is written in JavaScript. It's a backend service for the UI. It consists of different endpoints that perform a query to the database and serve the frontend with the data needed to display values and perform actions:

[12] https://picamera.readthedocs.io/en/release-1.13/
[13] https://scikit-image.org/
[14] https://pypi.org/project/pytesseract/

1) **getLatestReading.** Gets a document from the database that has the newest adding date. It is used to display the current reading.
2) **getConsumptionMonth.** Gets the water consumption of a given month. Month and year are added to the query to specify the data needed. If no specified info is added to the query, to-date consumption of the current month is given. The route is used to display the current to-date consumption of the month, the current balance of used water, the monthly consumption graph and to display the difference in consumption between the current and the previous month.
3) **getReadingMonth.** Gets the reading document of a given month. Month and year are added to the query to specify the data needed. If no specified info is added to the query, the data for the current month is given.
4) **getReadingsById.** Gets the reading document by a unique ID. It is used for testing purposes. Not used by the end-user.
5) **getReadings**. Gets all the reading documents in the Database. It is used for testing purposes. Not used by the end-user.
6) **sendReading**. Sends the reading to a given email address and changes the *sent* field value for the sent document from False to True. Takes in a body with fields of :{ id?; value?; email; month? }.
   Only the email field is required. Others are fully or partly optional - by default, the id value is used to identify the value and month from the reading that is being sent. The endpoint also accepts a free input value and month instead of the document. The latter is not used in this prototype but is rather an option for testing and further development.

4.2.3.3 water-meter-ui

Github link: https://github.com/peeterliik/water-meter-ui

This service is used to build the UI of the prototype application. The service consists of three pages and a sidebar with the corresponding navigational items:

1) **Dashboard**. The service uses one fixed-size card component to build the cards with changeable numbers, icons, and subtitles. A separate bigger fixed-size card component is used to display the monthly consumption graph. Data for the

components are fetched from the backend service endpoints. Node-fetch library is used for that[15] .

2) **Send readings.** Send readings page uses another card component which is a slightly modified version of the card component used to display value cards on the Dashboard page. It is slightly bigger and is comprised of an input field, text field, and an action button. The same component is used for displaying the recipient's email address card, sending a reading card, and schedule sending card.
3) **Settings**. The page uses another card component that has input and info fields for displaying and changing user data - name of the sender; home address; language. Card also has an action button that saves new info. Another card component is used to display a sample of the email that is being sent.

4.2.4 Main challenges

There were a number of challenges that emerged during the prototype build and development process. Here are some of them and how they were tackled.

- **Portable power source.** The initial plan was to build the digitiser in a portable form. This means that a power bank was considered a power supply. A power bank with the capacity of 2200mAh was tested to power the Raspberry Pi, but this proved to be unreasonable as it only lasted for a few hours with the camera module constantly powered on. Using a power bank as an energy resource for the digitiser might be a viable solution if, for one, the camera and/or Raspberry Pi would only power on for taking a picture, and for second - a power bank with a larger capacity would be used. In the end, the power supply option was dropped and the prototype was powered from a regular power outlet.
- **IR camera doesn't pick up red colour.** During the initial tests, it turned out that although the IR camera is perfect for taking pictures in a completely dark environment, it also had a downside of not picking up red colour and because of that the numbers after the decimal point were not visible on the picture. A solution for the prototype was to lay a colourized transparent film over the area after the decimal point to change the colour of the numbers. This proved to be a working solution.

[15] https://www.npmjs.com/package/node-fetch

- **Unstable camera placement.** A stable camera placement proved to be a vital aspect of getting consistent live data. The initial setup was a cardboard box with the Raspberry Pi together with the camera module taped on the side of the box. This was clearly unstable as the camera moved ever so slightly and this resulted in unstable picture quality - different focus and framing, which, in turn, made the automatic picture cropping and editing inaccurate. Additionally, the connection between the camera module and Raspberry Pi was not strongly fixed, and the movements of the camera resulted in the camera module cable detaching ever so slightly from the Pi's connector, which, in turn, froze the OS on the computer and the whole system had to be restarted. The solution was a stable camera tripod on which the Pi together with the camera module was attached. Lines were also drawn on the floor in order to maintain a more stable position throughout the live data tests.
- **Recognizing partly visible numbers**. The analogue counter uses rotating number wheels to form a display for the counter value. This means that at some point, the numbers on the wheels are inevitably not fully visible or not fully level, which in turn makes the number recognition part more difficult. This, in fact, was the main reason for readings not being recognisable and caused many faults. For this prototype, these faulty reads were discarded, and this meant that the live data feed was not stable at all times. The average percentage of faulty readings during the two week testing period proved to be about 60-70%. This varied a lot depending on the day. Data feed inconsistency was at most times at a toleratable level. Some days didn't give a valid reading, but for the whole two week period, there were still a number of valid readings to have enough data for the digitiser to work. The faulty readings percentage could be potentially lowered dramatically by testing out different settings for the pytesseract and different image manipulation settings.

5. Discussion

The following chapter discusses the possibilities of turning the prototype into a product and also the possibilities of using the same technology for other utility metering concepts.

5.1 Potential of turning the prototype into a product

This prototype was in operation in an apartment, digitising a mechanical water meter for a period of two weeks. It can be concluded from the experiment that the prototype is already a working substitute for an actual smart meter as it is giving the user the most significant benefits of the smart meter - automated reading reporting and a dashboard overview of the consumption history.

Still, the digitiser is a prototype and can not yet be considered a marketable product. Next, it will be discussed what should and could be improved in order to scale the prototype to a potential commercial product.

5.1.1 Software

The software for the prototype is written in a way that it could be scaled straight onto an actual product. Still, some adjustments are needed.

- **User management.** The prototype lacks the possibility to authenticate the user of the app. Also, there is no way to add a user, and only some of the user settings are currently changeable. For example, most importantly, the sender's email address can’t be changed. A middleware for user management should be added, and a database model for the users has to be implemented. The current code should be easily scalable for these tasks.
- **Cloud database storage.** For security reasons, currently, the database is stored on the Raspberry Pi. In order for the products database to be accessible to the end-users, the database should be stored in a cloud. The same database can be stored in all the major cloud database service providers. A secure user management middleware is a prerequisite for storing the database in a cloud.
- **UI and design improvement.** Together with the user management module, necessary design and UI improvements should also be implemented - most notably the whole login screen and additions to the user settings page.
- **Character recognition settings.** The percentage of valid readings could be improved. The faulty readings percentage varied a lot, but on average, during the two week period of testing, it was about 60-70%. Although tolerable and still functional, for the

actual product, the percentage of valid readings should be higher. One option is to do more in-depth testing in order to find the right character recognition settings and image manipulation parameters that are optimal for achieving a valid recognition.

- **(Optional) Camera operation logic.** Currently, the readings-recognition service doesn't use a very power efficient mode for taking the picture as it keeps the camera turned on all the time. Even during the time that the camera is not used. This could be changed into a more systematic power cycle to reduce the power consumption of the digitiser significantly.

5.1.2 Hardware

The hardware is the part that is lacking most when considering scaling the prototype to a product. Several difficult engineering tasks have to be taken into account
:

- **Compact size**. Although the prototype uses a compact size Raspberry Pi Zero computer with a small ZeroCam camera module, it is probably still a bit too big for the actual end product. Usually, the available space around the mechanical water meter is very limited. This in turn means that a different prototype and camera module should be used. Raspberry Pi has an even smaller size computer called Pico[16] available. A thorough research should be made in order to find the smallest available but affordable computer and camera to meet all the necessary requirements.
- **Custom case.** Currently, the computer and camera are attached to a camera tripod. This obviously isn't a suitable fit for a product. A custom case should be made to house the computer and the camera. Preferably something that would be attachable to the actual display of the mechanical water meter in order to have a universal fit and take into account the limited space available around the water meter.
- **Power consumption solution.** A suitable solution for power input should be engineered for the digitiser. It would be reasonable to use something that would take power from a regular power outlet as this would be most stable for the computer's work cycle. It has to be taken into account that usually there is no power line or outlet around the water meter. So, probably, a separate line has to be run to the water meter in order to power the digitiser.

[16] https://thepihut.com/collections/pico/products/raspberry-pi-pico

5.2 Using OCR for digitising analogue counters

This experiment of digitising a mechanical water meter with a custom-built digitiser showed that a similar technology and software could be used for digitising any sort of analogue metering device. The prototype uses optical character recognition (OCR) software for turning the analogue data into a machine-readable form and this could be used universally for example with analogue electricity or a gas metering counter.

The software and hardware would only need some minimal adjustments in order to work with a different sort of analogue utility metering device.

Additionally, the technology could also be used for different sorts of analogue counters on the industrial level. This could prove to be an affordable and fast intermediate solution for an industry that still uses mechanical counters for whatever purpose - on a consumption metering device, production line item counter etc.

Conclusion

The aim of the thesis was to build a live data prototype of a digitiser that would add the main benefits of a smart meter, remote reading, and consumption history to a regular mechanical analogue water meter. Thus, serving as a viable low-cost intermediate substitute to an actual smart meter.

The key aspect, turning analogue metering data into digital machine-readable form, was solved by using the optical character recognition tool Tesseract on a photograph of the value of reading. Despite a significant percentage of faulty readings, the method still proved to work as enough valid and usable live digital data was produced from the mechanical water meter.

The usability of the data was tested by building a personal water consumption dashboard together with the options for the user to send out readings and set automatic readings

reporting schedule. Modern JavaScript runtime Node.js together with React.js, was used to build the application.

Proof of concept also showed the potential of turning the prototype into a product that could serve as an intermediate substitute for an actual smart meter. The universal nature of the digitiser prototype also gave reason to consider the same technology for digitising other mechanical utility metering devices' output.

References

- Riigi Teataja. (2019). *Grid Code–Riigi Teataja*. Retrieved May 9, 2022, from *https://www.riigiteataja.ee/en/eli/ee/VV/reg/519122019003/consolide*
- Tallinna Vesi. (2022). *Aastaaruanne 2021*. Retrieved May 9, 2022, from *https://tallinnavesi.ee/wp-content/uploads/2022/04/Tallinna-Vesi-2021-auditeeritud-aastaaruanne-1.pdf*
- Pärnu Vesi. (2021). *Aastaaruanne 2020*. Retrieved May 9, 2022, from *https://www.parnuvesi.ee/wp-content/uploads/2021/05/2020__Aruanne.pdf*
- Pott, T. (2019). *Tulevaks aastaks tuleb gaasiarvestid välja vahetada*. Eesti Rahvusringhääling | ERR. Retrieved May 9, 2022, from *https://www.err.ee/1002491/tulevaks-aastaks-tuleb-gaasiarvestid-valja-vahetada*
- Eesti Vee-ettevõtete Liit. (2022). Eesti Vee-ettevõtete Liit homepage. Retrieved May 9, 2022, from *https://evel.ee/*
- Eesti Vee-ettevõtete Liit. (2022). Eesti Vee-ettevõtete Liit - kaugloetavad arvestid. Retrieved May 9, 2022, from https://evel.ee/teabepank/kaugloetavad-veearvestid
- Cook, B., Gazzano, J., Gunay, Z., Hiller, L., Mahajan, S., Taskan, A., & Vilogorac, S. (2012). The smart meter and a smarter consumer: quantifying the benefits of smart meter implementation in the United States. *Chemistry Central journal*, *6 Suppl 1*(Suppl 1), S5. *https://doi.org/10.1186/1752-153X-6-S1-S5*
- Monica, P & Crainic, M. (2012). *A short history of residential water meters. Part 1, mechanical water meters with moving parts. Installations for Buildings and Ambiental Comfort Conference XXI- edition*
  Timisoara - ROMANIA 18-20April 2012, pp. 27-35
- Eesti Vee-ettevõtete Liit. (2022). Eesti Vee-ettevõtete Liit homepage. Retrieved May 9, 2022, from *https://evel.ee/*
- Kamstrup (2021). *floqIQ2200 - Data Sheet. Retrieved May 9, 2022, from https://www.kamstrup.com/en-en/water-solutions/meters-devices/meters/flowiq-2200-eu/documents*
- Cagan, M (2014). Silicon Valley Product Group: Flavors of Prototype. Retrieved May 9, 2022, from *https://www.svpg.com/flavors-of-prototypes/*
- Snow, J (2017). *Smart Cities Dive: High cost of smart water meters keeps adoption rates low.* Retrieved May 9, 2022, from

https://www.smartcitiesdive.com/news/high-cost-of-smart-water-meters-keeps-adoption-rates-low/443905/

- Küttemaailm OÜ (2022). *Axioma Ultrasonic metering devices*. Retrieved May 9, 2022, from *https://optimatic.ee/wp-content/uploads/2022/02/axioma_ultrasonic_metering_devices_2022.xlsx*
- IBM Cloud Education (2022). *What Is Optical Character Recognition (OCR)?,* Retrieved May 10, 2022, from https://www.ibm.com/cloud/blog/optical-character-recognition
- Raspberry Pi (n.d.) *About us*. Retrieved May 10, 2022, from *https://www.raspberrypi.com/about/*
- Tesseract OCR (n.d.). *Tesseract Open Source (OCR) main repository*. Retrieved May 10, 2022, from *https://github.com/tesseract-ocr/tesseract*
- Logicmonitor (n.d.). *What Are Microservices and Why Use Them?* Retrieved May 10, 2022, from *https://www.logicmonitor.com/blog/what-are-microservices*
- Raspberry Pi OS (n.d.). Raspberry Pi OS documentation. Retrieved May 10, 2022, from *https://www.raspberrypi.com/documentation/computers/os.html*
- Python Wiki (n.d.). Beginners Guide/Overview. Retrieved May 10, 2022, from *https://wiki.python.org/moin/BeginnersGuide/Overview*
- PYPL (2022). *PYPL PopularitY of Programming Language.* Retrieved May 10. 2022, from *https://pypl.github.io/PYPL.html*
- Python Developer (2022). *Brainstation - Who Uses Python Today?* Retrieved May 10, 2022, from *https://brainstation.io/career-guides/who-uses-python-today*
- Hack Reactor (2021). *What is Javascript used for?* Retrieved May 10, 2022, from *https://www.hackreactor.com/blog/what-is-javascript-used-for*
- Node.js (n.d.). *About Node.js*. Retrieved May 10, 2022, from *https://nodejs.org/en/about/*
- React (n.d.) *Frontpag*e. Retrieved May 10, 2022, from *https://reactjs.org/*
- MongoDB (n.d.) *Why use MongoDB and When to use it?* Retrieved May 10, 2022, from *https://www.mongodb.com/why-use-mongodb*
- Kamstrup. (2022). *Kamstrup Multical 21*. [Photograph]. Kamstrup. *https://koce1-kamstrup.ocecdn.oraclecloud.com/content/published/api/v1.1/assets/CONT002CAEEC8EE34730AC905878CF26FC88/XXXXXX660x495XXXXXX/MC21_GB_Front_110mm.png?format=webp&type=customrendition&channelToken=ed241bbb18f444908a8fc9ed97ca5d5b*

- The NoBroker Times. (2022). *Water Meter*. [Photograph]. The NoBroker Times. https://www.nobroker.in/blog/wp-content/uploads/2020/07/water-meter-NoBrokerBlog.png
- Raspberry Pi. (2022). *Raspberry Pi Zero 2 W*. [Photograph]. Raspberry Pi. *https://assets.raspberrypi.com/static/51035ec4c2f8f630b3d26c32e90c93f1/2b8d7/zero2-hero.webp*
- The PiHut. (2022). *ZeroCam NightVision for Raspberry Pi*. [Photograph].The Pi Hut. *https://cdn.shopify.com/s/files/1/0176/3274/products/zerocam-nightvision-for-raspberry-pi-the-pi-hut-102351-23122227281_600x.jpg?v=1646063644*

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, _____Peeter Liik_____,
(autori nimi)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

_____ Live data prototype of a mechanical water-meter digitiser
_____
_____
_____,
(lõputöö pealkiri)

mille juhendajad on _____Ahmed Aboumalwa ja Heili Orav_____,
(juhendaja nimi)

reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 4.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Peeter Liik
18.05.2022