Smart Water Meter using AI-on-the-edge

By: Shreya Gupta (29942) and Somaya Ahmadian (30551)

1. Introduction

In the recent past, the demand for freshwater worldwide has increased largely due to the world's growing population, as well as changing lifestyles and eating habits that have been associated with higher water consumption [1]. The increase is more pronounced in urban settings which normally have higher population densities in addition to production industries that typically consume large amounts of water [1]. Water efficiency implies less water consumption and searching for an alternative of conventional water meters to measure the quantity and quality of water [2].

Water meters are used for urban water management, especially for billing purposes [1]. The relatively recent Smart Water Metering (SWM) technology provides high resolution and frequent water consumption data which can be used to improve feedback to consumers and thus enhance water conservation and management [1].

Furthermore, a SWM system will make users mindful of their water consumption and help them to reduce their water usage. At the same time, users will be alerted to abnormal water usage to reduce water loss [3].

In this project, we employed the ESP32CAM and the "AI-on-the-edge" technology to develop a Smart Water Meter system. The idea of the project is to use an ESP32CAM, which is a camera aid on a microcontroller, that captures periodic images of the water meter. The meter reading is then detected using AI, and the data is transmitted to a Home Assistant container in Docker through MQTT (Message Queuing Telemetry Transport) where it can be further analysed and set up according to user's requirements.

2. Materials

2.1. ESP32

The ESP32 is a series of chip microcontrollers developed by Espressif. You can get an ESP32 starting at $6, which makes it easily accessible to the general public. It consumes very little power compared to other microcontrollers, and supports low-power mode states like deep sleep.

The ESP32 microcontroller can easily connect to a Wi-Fi network (station mode), or create its own wireless network so other devices can connect to it. This is essential for IoT and Home Automation projects. We can have multiple devices communicating with each other using their Wi-Fi capabilities.

It has its own built-in flash memory and supports external SPI flash and PSRAM for additional memory as well.

It also supports Bluetooth classic and Bluetooth Low Energy (BLE), which is useful for a wide variety of IoT applications. For example, it is compatible with the Arduino, so we can program the ESP32 in

Arduino IDE using ESP8266, ArduinoUno or other microcontrollers of such capabilities.

https://randomnerdtutorials.com/getting-started-with-esp32/

ESP32CAM The ESP32-CAM is the microcontroller that we used in this project that comes with a detachable camera aid. It costs approximately $10. Besides the OV2640 camera, and several GPIOs to connect peripherals, it also features a microSD card slot that can be useful to store images taken with the camera or to store files to serve to clients. For the purpose of our project we have used an ESP32Cam by AIThinker.

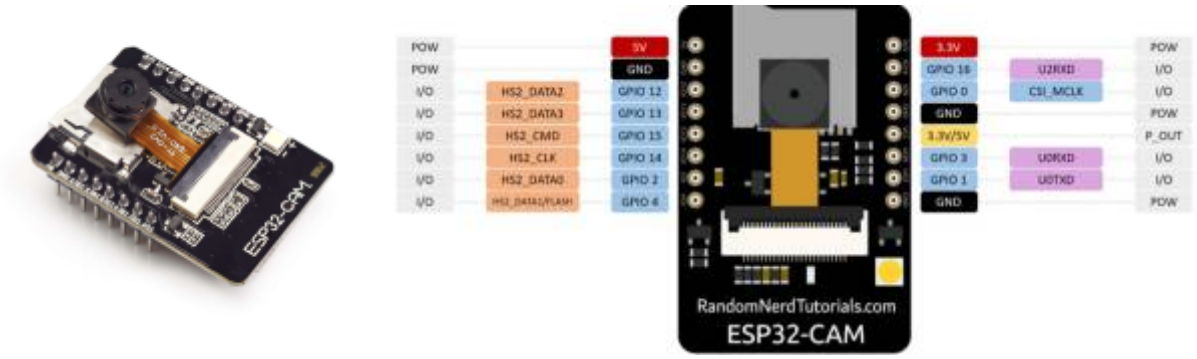
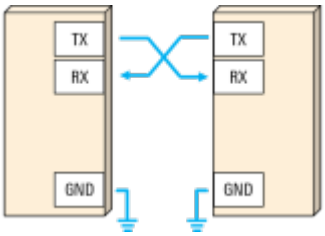


Figure 1: ESP32CAM https://www.makershop.de/plattformen/nodemcu/esp32-cam-board/

Source: https://randomnerdtutorials.com/esp32-cam

2.2. UartSBee

UART stands for universal asynchronous receiver/transmitter and is a simple, two-wire protocol for



exchanging serial data.

Asynchronous means no shared clock, so for UART to work, the same bit or baud rate must be configured on both sides of the connection. Start and stop bits are used to indicate where user data begins and ends, or to “frame” the data.The idle state is where no data is being transmitted. An optional parity bit can be used to detect single bit errors.

UART is still a widely used serial data protocol but has been replaced in some applications by technologies such as SPI, I2C, USB, and Ethernet in recent years.

Source:
https://www.rohde-schwarz.com/us/products/test-and-measurement/essentials-test-equipment/digital-oscilloscopes/understanding-uart_254524.html#gallery-8

2.3. 16 GB MicroSD card

Micro SD cards are frequently employed to expand storage when the built-in memory is insufficient. In this project, a MicroSD card is utilised with the ESP32CAM to store data, expanding memory for images captured by the camera, including water meter counts and also to help install the firmware on the microcontroller.

2.4. Jumper cables

Jumper cables, which are insulated wires, establish connections between various components in this project. We used several different types of jumper wires to create the necessary links, connecting them to header pins on the ESP32CAM and UART module.

2.5. USB cable

USB, in full universal serial bus, is a technology used to connect computers with peripheral devices [7]. In our project, we established a connection between the ESP32CAM and our computer using a USB cable. The primary use was for programming the ESP32CAM through the UART module, allowing us to upload and update the code to its microcontroller. Additionally, the USB cable provided power to the system directly from our computer, eliminating the need for an external power supply while working on the project.

2.6. Water meter

A water meter is an instrument used to measure the water consumption in a building connected to the municipal water supply. In residential settings, the meter reading ensures accurate water billing

and can detect any potential leaks within the home.

Source: https://www.freshwatersystems.com/blogs/blog/what-is-a-water-meter-and-how-do-i-read-one

2.7. Casing

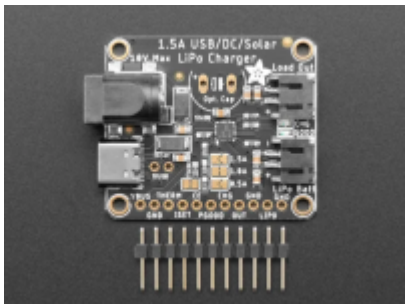
The ESP32 casing serves as a protective enclosure to securely hold the ESP32CAM and its camera module. For this project, we downloaded an existing design from https://www.thingiverse.com/thing:4845508 and used the available 3D printer in the Green FabLab to fabricate it.

2.8. LiPO Battery

A 3.7V LiPo battery was used to supply power to our SWM system. It can be connected to a LiPo charger/adapter for recharging and also to establish connection (without manually removing the wires) through a voltage regulator (for optimal voltage output) to the camera module.

2.9. LiPo charger/adapter

An Adafruit 1.5A USB/DC/Solar LiPo Charger was used to establish the connection of the power supply system and for the purpose of recharging the battery as well as providing other options of power supply such as a solar panel or USB connection to a power source.



2.10. Step-up/Step-down Voltage Regulator

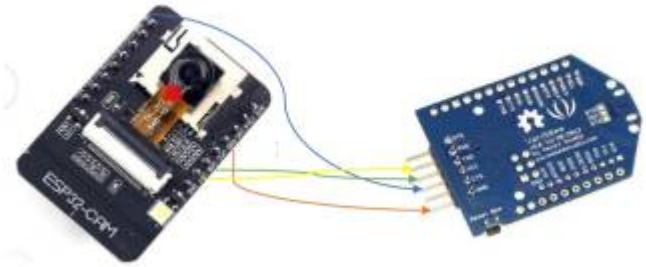
A Pololu Step-up/Step-down Voltage Regulator was used to boost the voltage output for the camera module. This piece of equipment needed to be soldered to be able to establish connections using the

jumper cables.

3. Method

3.1. Wiring

The initial step involved connecting the ESP32CAM and UART module, using jumper cables. This ensured the proper flow of data between the devices.



The pin connection	
ESP32CAM	UartSBee
Ground (GND)	GND
5V	VCC (5V)
U0R	TX
U0T	RX
GPIO 0	GND

Note: GPIO 0 needs to be connected to GND, to make sure the ESP32 does not boot normally but starts in programming (flashing) mode after being powered on.

3.2. Firmware Installation

Manually: After completing the setup and ensuring all connections were made, we plugged in the USB connection through the UartSBee and started Terminal (on Mac). There, we installed the necessary Python library (esptool), which allows us to modify the firmware of the ESP32. To determine the COM port of the device, we utilized Arduino software and checked under “Tool” and “Port.”

Next, we navigated to the firmware folder on the hard disk and followed the installation instructions from the AI_on_the_edge documentation[4]. We initiated the process with the command “esptool erase flash.” In our case, we had to rename the esptool command to esptool.py on Mac. We then executed the second command to flash the bootloader firmware and partitions.

```
esptool erase_flash
esptool write_flash 0x01000 bootloader.bin 0x08000 partitions.bin 0x10000 firmware.bin
```

The next step, was to initialize the Ai-on-the-edge through Arduino's serial monitor. It showed us that the connection was established over Wi-fi for the camera module and it provided us with the IP address to set up the AI interface.

Note: Occasionally, the ESP did not connect, requiring us to press the reset button or briefly disconnect the power cable. We also disconnected the jumper cable at pin “IO0” and briefly interrupted the power supply. This allowed the ESP32CAM to boot the new firmware, and the LED began flashing after a few seconds.

Web-Installer: On some occasions, the camera module could not establish a connection with the wifi or provide us with the URL to the AI-on-the-edge webpage. In those cases, we opted to use the web-installer service mentioned in the documentation provided by jomjol[4]. We could easily connect to the right port, install the firmware and then use the “Logs & Console” option to lead us further (Note: After the installation, the GPIO to GND connection needs to be disconnected to take the module off bootloader mode. Additionally, when the connection didn't work, the transmission&receiver pins were disconnected as well in case the SD card confused the connection). It provided us the IP address of the AI service.

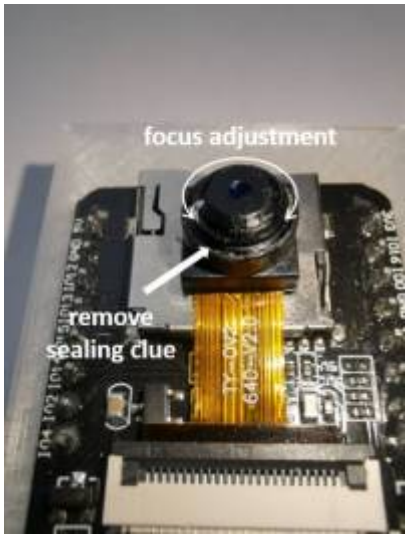
3.3. SD Card

Before flashing the firmware, the SD card was inserted into the computer, verifying that it had a “Master Boot Record partition formatted” as FAT32. We proceeded to copy the downloaded contents onto the SD card. Then, we edited the “wlan.ini” file, adding our Wi-Fi “SSID” and “password”, and saved the changes before ejecting the SD card.

Next, we inserted the SD card into the ESP32CAM. After obtaining the IP address, we entered it into the browser, which allowed us to access the AI on the edge setup interface.

3.4. Camera focus

Before proceeding further, we needed to adjust the focus ring of the ESP32 camera to achieve the desired image resolution. For that, we carefully removed the glue from the lens rotator using a knife. The ring was then rotated clockwise until the image became sharp and clear, enabling us to achieve the optimal focus for the readings.



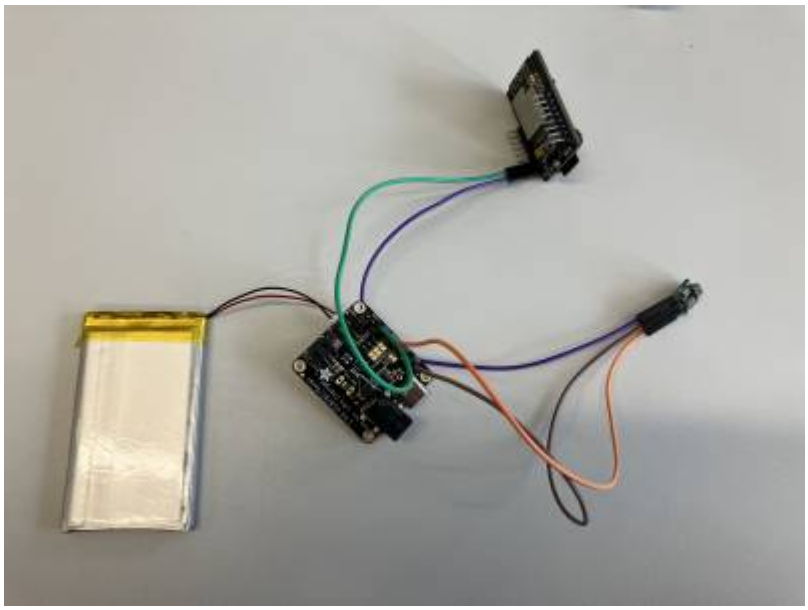
3.6. Casing

For the protective casing of the ESP32CAM module and water meter, we opted for a pre-designed 3D model mount[6]. Utilizing Fusion 360 software for design preparation and Cura for slicing the 3D design and creating the file to be sent to the printer, we successfully 3D printed the casing (requiring approximately 9 hours to complete). To ensure a precise fit on the water meter, we meticulously adjusted the mount by cutting a small part, allowing it to perfectly accommodate the water meter's dimensions. The modified casing securely encased the ESP32CAM, providing essential protection for our project.

3.7 Power Supply

After installing the firmware on the ESP32Cam and setting up the AI-on-the-edge interface (part of results section), we had to ensure that we will be able to provide continuous power supply to our SWM. Hence, we opted to use a LiPo battery (3.7 V) for the power supply to our camera module. This battery is rechargeable through a Universal LiPo charger that also has ports for USB and solar panel connections for charging. This charger was connected as an adapter to connect the battery to Step-up/Step-Down Voltage Regulator (in our case it is a Step-Up voltage regulator supplying 5V output) to the camera module. The V-out of the regulator was connected to the camera module's 5V pin, while

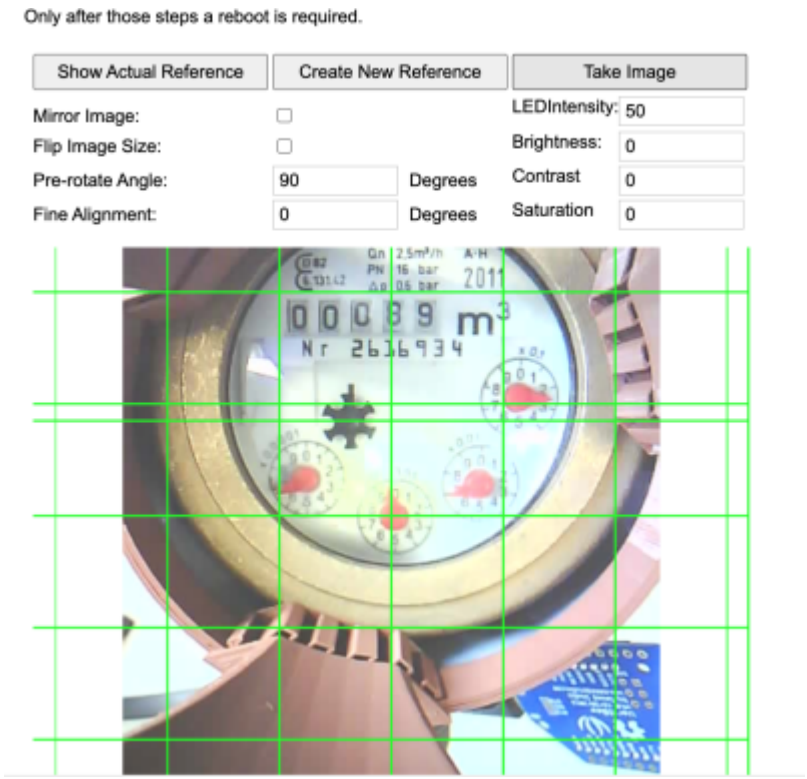
the GND was connected to the GND of the charger and the camera module's GND also was connected to the other GND of the charger. The V-in of the regulator was connected to the Out pin of the charger.



4. Results

4.1. Digitizer (Initial Setup)

After accessing the IP address, the AI-on-the-edge setup webpage appeared. The first step was to create a new reference image, so we clicked on “Create a new reference image.” Initially, the focus was not ideal, so we adjusted the camera lens until achieving the sharpest focus possible.

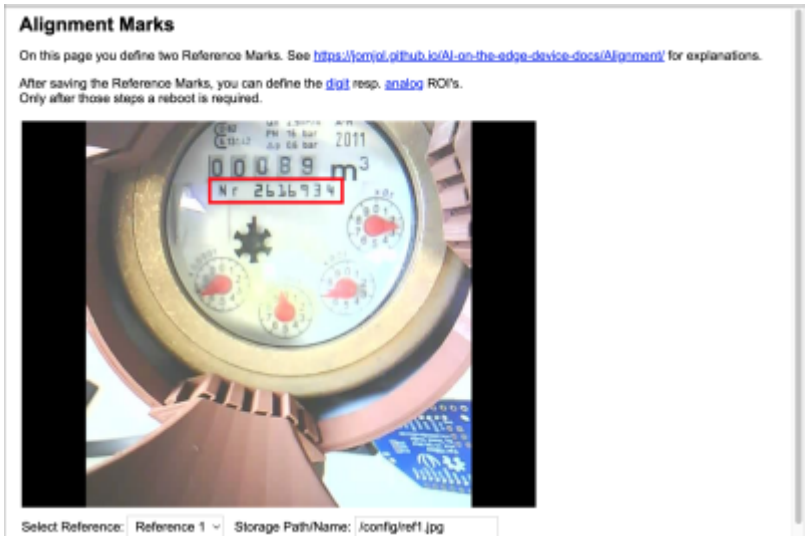


However, we encountered an issue with the ESP32's built-in LED, as its flashing caused unwanted reflections on the glass of the water meter, making the AI-based readout challenging. To address this, we decided to cover the LED with a piece of tape, which provided better control over reflections and improved the accuracy of the readings.

Note: In our specific case, we didn't require the LED since the experiment took place during the day, providing sufficient natural light. However, in real-life situations where water meters are often located in dark places like basements, adequate lighting is essential for capturing clear images. To address this, one option is to set up the system with regular LEDs or, even better, use infrared LEDs. Infrared LEDs offer the advantage of remaining invisible to the human eye while providing illumination for the camera to capture images. However, to use infrared LEDs, it is necessary to remove the infrared filter from the camera lens to enable the camera to detect infrared light accurately.Another option is to use a light diffusing material (when using the camera's built-in LED) to cover the empty parts of the encasing such that it reflects an optimum amount of light that can make it easy for the camera to read under all conditions. An example of such a material is baking paper which can be wrapped around the casing, providing it a good amount of diffused lighting through the white buttery surface, while the dark surface of the casing absorbs the rest.

4.2. Image analysis

In this step, we needed to ensure perfect alignment of the numbers. To achieve this, we made adjustments such as “pre rotate angle” and “Fine alignment.” Once the alignment was satisfactory, we clicked on “Update image.” Moving forward to the next step, we were prompted to select two reference points on the water meter. After selecting the references, we saved our changes.



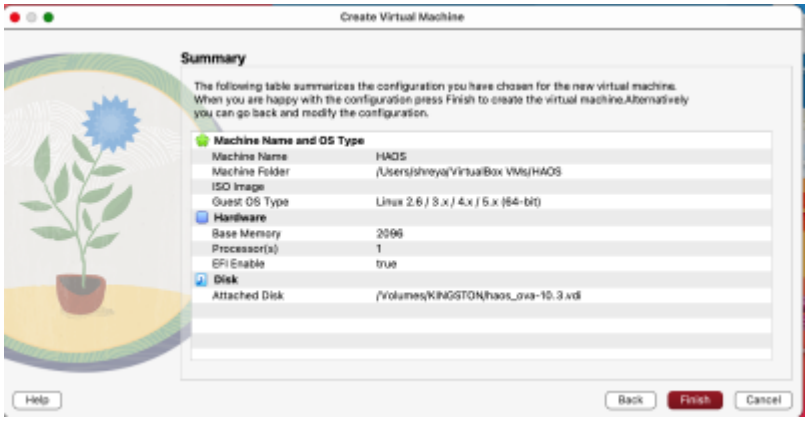
Following this, we carefully opted for digits that were precise and centred as much as possible in the “Define Digits” section. Using the “Number” selector, we defined single digits and introduced new numbers. We could conveniently switch between different digits using the drop-down menu labelled “ROIx” (Region Of Interest).

To adjust the positions of the ROIs within the reading, we used the options “Move Next” or “Move Previous.” Finally, we saved all our changes with “Save all to Config.ini.”Since our water meter lacked analog information, we unchecked the next step.

In the subsequent stage, we focused on the “General Configuration Parameters” section to access more settings. An essential part of this was navigating to the “MQTT” settings.

4.3. MQTT in Home Assistant

Further in the process, using the Oracle VirtualBox, a virtual machine was setup to run Home Assistant in a Linux based environment. Following the documentation on the Home Assistant website, the configuration was complete [8]. Once the setup was finished, Mosquitto Broker was installed as an Add-on into the Home Assistant. We had used the digitizer in the AI-on-the-edge initialisation to set up the MQTT credentials to be used for this process.



5. Discussion

While the SWM system is an inexpensive way to monitor household water consumption and the setup is relatively straightforward, the whole process is filled with tiny technological hurdles that present themselves at different stages. There were several instances where the camera module just stopped functioning, even after establishing all sorts of protocols and measures to relieve the process of any

hiccups, the module stopped functioning in the end. Another module had to be used to replace the old one.

Additionally, the Virtual Machine installation and operation in Docker (which is not entirely possible due to operating system limitations), establishing MQTT broker and using Home Assistant in a Docker container were the most perplexing steps to familiarise oneself with, without having any prior introduction to such concepts. Furthermore, prior to the installation of MQTT add-on in Home Assistant, the virtual machine needed to be run through VirtualBox which did succeed fully when using the localhost port. This halted the installation of the Mosquitto Broker in the HAOS created and limiting any further steps to proceed in the project.

Although, the initial learning material helped us form the basis of establishing the connections, physical setup and conceptual understanding of the system and its components.Further training and information material would immensely help in understanding and dealing with the software components after the instructions on the hardware.

6. References

[1] T Randall, R Koech. (2019). Smart Water Metering Technology for Water Management in Urban Areas. Water e-Journal, Volume 4 No 1 2019. ISSN 2206-1991.
[2] Raad AL-Madhrahi, Jiwa Abdullah, and Nayef.A.M. “An Efficient IoT-based Smart Water Meter System for Smart City Environment.” International Journal of Advanced Computer Science and Applications (IJACSA), vol. 12, no. 8, 2021, pp. 420-422. www.ijacsa.thesai.org.
[3] Abu-Mahfouz, A.M. and Steyn, L.P. (2015). Smart water meter system for user-centric consumption measurement. In Proceedings of the IEEE 13th International Conference on Industrial Informatics (INDIN), 22-24 July 2015, Cambridge, UK. DOI: 10.1109/INDIN.2015.7281870.
[4]https://jomjol.github.io/AI-on-the-edge-device-docs/
[5]https://www.libe.net/en-watermeter
[6]https://www.thingiverse.com/thing:4845508
[7]https://www.britannica.com/technology/USB
[8]https://www.home-assistant.io/installation/macos