Group B: Matsvei Yankouski, Stas Besliu, Demid Dabizha

Smart metering: water meter with computer vision

Introduction

Water meter devices play a crucial role in measuring and monitoring water consumption. These devices are designed to accurately quantify the amount of water consumed by residential, commercial, and industrial properties. By providing accurate and reliable data on water usage, water meter devices enable efficient billing and help detect potential leaks or abnormal usage patterns. With advancements in technology, modern water meter devices are equipped with smart features, allowing for remote monitoring, data analysis, and real-time alerts. Such devices not only promote water conservation and sustainability but also empower users with valuable insights to optimize their water usage and contribute to a more responsible and efficient water management system.

The development of a smart water meter system incorporating a camera for reading consumption numbers and transmitting them to users opens up new possibilities for water monitoring and analysis. By integrating a camera into the metering process, the system can capture accurate readings of consumption numbers with minimal manual intervention. These readings are then transmitted to the users, enabling them to access real-time data on their water usage. This empowers users to analyze their consumption patterns, identify trends, and make informed decisions regarding their water usage. With the ability to generate statistics based on this data, users can gain insights into their water consumption habits, set conservation goals, and take proactive measures to reduce waste. Ultimately, the combination of a camera-enabled smart water meter system and user data analysis contributes to promoting water efficiency and fostering a more sustainable approach to water management.

Materials and Methods

For this project we used:

1) A domestic water meter, jumper cables

2) ESP32-cam module

(Specifications could be found here: link)

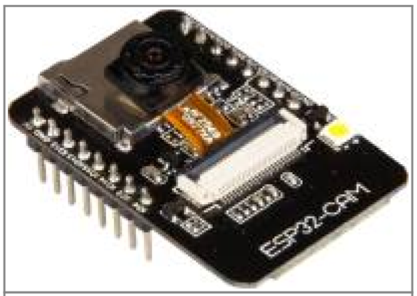


Fig.1: ESP32-CAM

The ESP32-CAM module is a versatile and compact development board that combines an ESP32 microcontroller with a camera module. It offers a wide range of functionalities, including Wi-Fi and Bluetooth connectivity, making it an excellent choice for projects that require image capture and transmission. With its built-in camera, the ESP32-CAM enables users to capture and stream images or videos in real-time, making it ideal for applications such as surveillance systems, IoT projects, or even remote monitoring. Its small form factor and rich feature set make it a popular choice among developers and enthusiasts alike.

3) UartSBee v5.0 module

(Specifications could be found here: link)

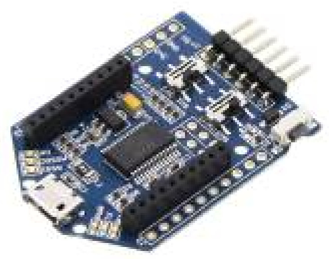


Fig.2: UARTSBee

The UARTSBee v5.0 module is a handy and user-friendly communication module that facilitates serial communication between devices. Equipped with a USB-to-serial chip, it enables easy connection and data exchange between a computer and various electronic devices or microcontrollers. Its compact design and robust functionality make it an essential tool for debugging and programming microcontrollers, as well as for establishing reliable serial communication in DIY electronics projects. With its versatility and simplicity, the UARTSBee v5.0 module serves as a reliable bridge for transmitting data over UART interfaces, simplifying the process of communication in numerous applications.

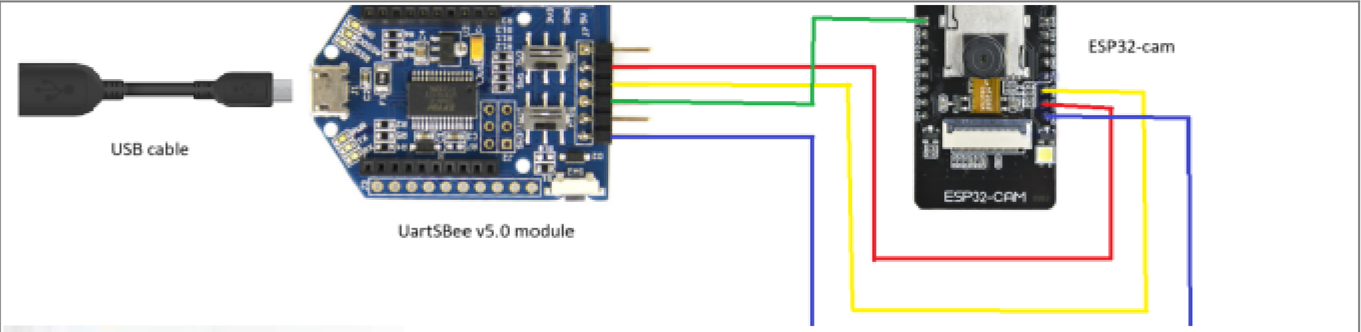
4) USB A cable to micro USB cable

USB (Universal Serial Bus) is a widely used interface technology that enables the connection and communication between electronic devices. It simplifies the process of data transfer, device charging, and peripheral connectivity. The USB standard consists of various types of connectors, including Type-A, Type-B, and Type-C, each serving specific purposes. When a USB device is connected to a host device, such as a computer or smartphone, it establishes a communication link through a series of protocols. These protocols define the data transfer speed, power delivery, and the type of data being transferred. The USB interface provides a plug-and-play functionality, allowing devices to be hot-swapped without requiring a system restart. With its versatility, ease of use, and widespread compatibility, USB has become an indispensable technology in the modern digital ecosystem.

5) MicroSD card

SD (Secure Digital) card is a small, portable storage device commonly used in cameras, smartphones, and other electronic devices. It utilizes flash memory technology to store and retrieve digital data. The SD card consists of non-volatile memory chips, a controller, and contact pins for connectivity. When inserted into a compatible device, such as a camera, the SD card establishes a physical connection, allowing data to be read from or written to the memory. The controller manages the data transfer and ensures the integrity of the stored information. SD cards are available in different capacities and speed classes, providing options for various storage needs. With their compact size, durability, and high data transfer rates, SD cards offer a convenient and reliable solution for expanding storage capacity and transferring data between devices.

Our final setup looks like that:



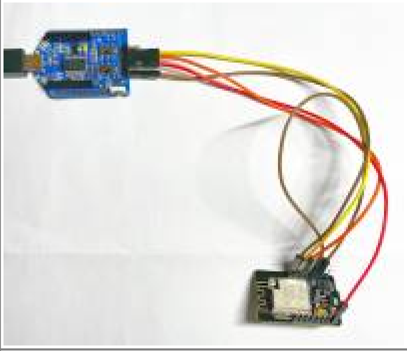


Fig.3: Theoretical + Physical Setup

For the setup we had to perform a couple of steps:

- Prepare the camera focus

The initial setup of the camera had glue on the lens protector and on the lens rotator.^^

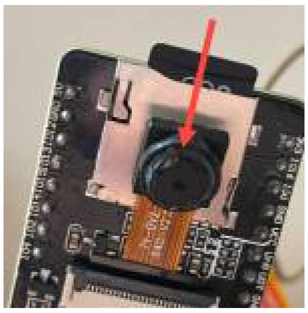


Fig.4: Broken camera filter

So to make a proper focus we had to remove this glue or simply just to loosen the rotator. We made it using a knife and pliers. Unfortunately, the rotator is very easy to break so we broke it but it never bothered the focus.

- Installing the firmware on the SD card and on our computer

For this step we used the AI-on-the-edge project from Jomjol: link to the github of Jomjol

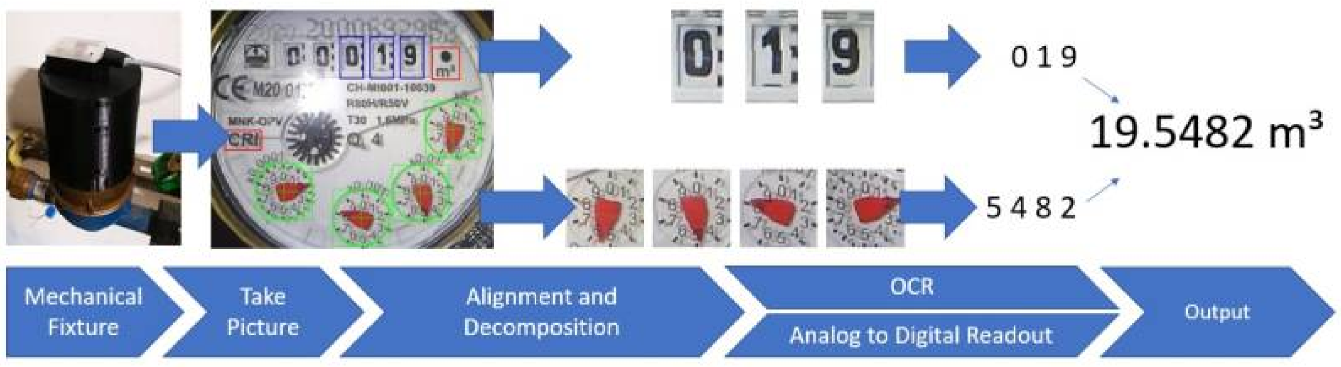
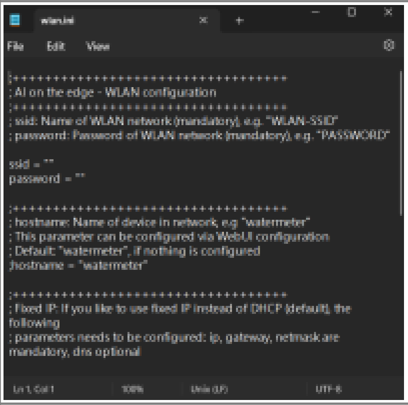


Fig.5: Working Principle



```
;++++++++++++++++++++++++++++++++++
; AI on the edge - WLAN configuration
;++++++++++++++++++++++++++++++++++
; ssid: Name of WLAN network (mandatory), e.g. "WLAN-SSID"
; password: Password of WLAN network (mandatory), e.g. "PASSWORD"

ssid = ""
password = ""

;++++++++++++++++++++++++++++++++++
; hostname: Name of device in network, e.g "watermeter"
; This parameter can be configured via WebUI configuration
; Default: "watermeter", if nothing is configured
;hostname = "watermeter"

;++++++++++++++++++++++++++++++++++
; Fixed IP: If you like to use fixed IP instead of DHCP (default), the
following
; parameters needs to be configured: ip, gateway, netmask are
mandatory, dns optional
```

Fig.6: SD Card HTML File configuration

The project gives us a possibility to make our own computer vision system to interact with the water meter. The key features of the project are:

1. Tensorflow Lite (TFlite) integration - including easy to use wrapper
2. Inline Image processing (feature detection, alignment, ROI extraction)
3. Small and cheap device (3×4.5×2 cm³, < 10 EUR)
4. camera and illumination integrated
5. Web surface to administrate and control
6. OTA-Interface to update directly through the web interface
7. Full integration into Homeassistant
8. Support for Influx DB 1
9. MQTT
10. REST API

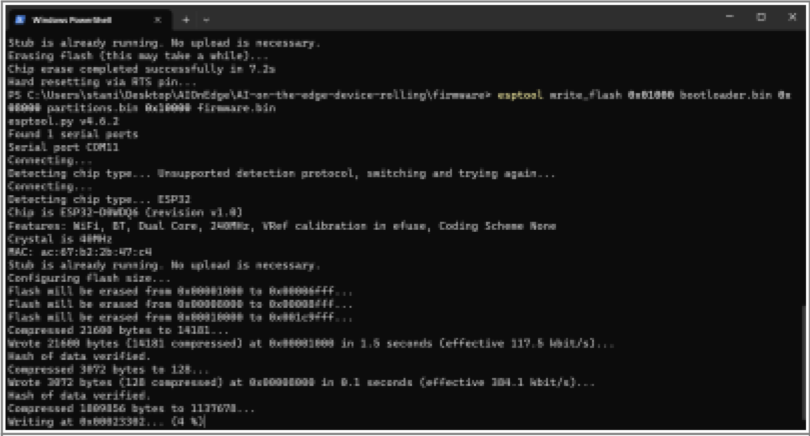
The initial step of the installation was to download the ZIP archive (the whole github repository). Then we just had to simply follow the instructions. After downloading the repository we copied the SD card folder to our SD card that we used for the camera setup. Inside we could find a WLAN.ini file where we had to insert our wifi credentials in order to have our camera connected to our wifi. The SD card we use should not be bigger then 16Gb and should have FAT32 format. Then we have to perform a couple of steps to set up our camera. For this task we navigate into MS PowerShell. We also have to have an installed Python of latest version. Here are the steps:

1. Open a powershell window
2. Install esptool library using “pip install esptool” command (it is possible that it will not work so maybe “python -m pip install esptool” or “pip3 install esptool” will work
3. Navigate to the firmware folder (for us it was cd “C:\Users\stani\Desktop\AIOnEdge\AI-on-the-edge-device-rolling\firmware”)
4. Connect the ESP32 module and perform the following two commands:

code.py

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

Then we have to wait a little bit until the code will be transferred into the ESP32 cam:



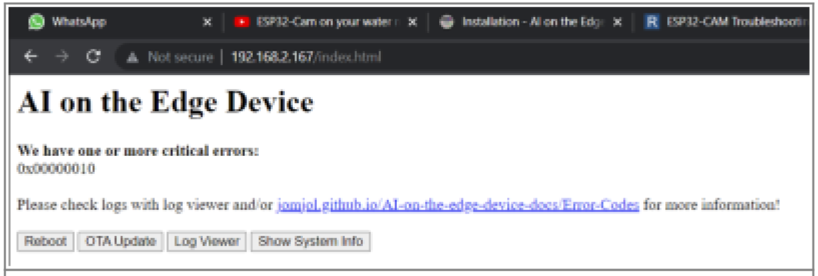
```
Stub is already running. No upload is necessary.
Erasing flash (this may take a while)...
Chip erase completed successfully in 7.2s
Hard resetting via RTS pin...
PS C:\Users\stani\Desktop\AIOnEdge\AI-on-the-edge-device-rolling\firmware> esptool write_flash 0x01000 bootloader.bin 0x
08000 partitions.bin 0x10000 firmware.bin
esptool.py v4.6.2
Found 1 serial ports
Serial port COM11
Connecting...
Detecting chip type... Unsupported detection protocol, switching and trying again...
Connecting...
Detecting chip type... ESP32
Chip is ESP32-D0WDQ6 (revision v1.0)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: ac:67:b2:2b:47:c4
Stub is already running. No upload is necessary.
Configuring flash size...
Flash will be erased from 0x00001000 to 0x00006fff...
Flash will be erased from 0x00008000 to 0x00008fff...
Flash will be erased from 0x00010000 to 0x001c9fff...
Compressed 21600 bytes to 14181...
Wrote 21600 bytes (14181 compressed) at 0x00001000 in 1.5 seconds (effective 117.5 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 128...
Wrote 3072 bytes (128 compressed) at 0x00008000 in 0.1 seconds (effective 304.1 kbit/s)...
Hash of data verified.
Compressed 1809856 bytes to 1137678...
Writing at 0x00023302... (4 %)
```

Fig.7: ESP32-CAM response

After performing these steps the ESP32 module should start to blink. If it blinks 4 times and then stops for a while then everything is ok and we can check it in Serial Monitor.

Results

After successfully installing the software we can move to the step of setting the camera up. For this purpose we use the ip address we can obtain from serial monitor info. For us it was 192.168.2.167, but it can differ from time to time. Immediately we had a problem of not showing the window:


WhatsApp
ESP32-Cam on your water
Installation - AI on the Edg
ESP32-CAM Troubleshootin
Not secure | 192.168.2.167/index.html

AI on the Edge Device

We have one or more critical errors:
0x00000010

Please check logs with log viewer and/or jomjol.github.io/AI-on-the-edge-device-docs/Error-Codes for more information!

Fig.8: Welcome HTML Page

We figured out that we had a problem with our wifi connection and it was repeating from time to time. We found a solution and it is to connect the camera to the mobile hotspot but later the problem repeated. So we think that just reconnecting helps but not for a long time.

After dealing with the mistakes we got from the ESP32 module we finally got to install the Wi-Fi and the page open successfully. After opening the ip address we got the information about the project and we can proceed further to the configuration process.

Firstly we had to make a clear reference image. We can simply do this following the instructions. The image is needed for the system to know where to find the values.

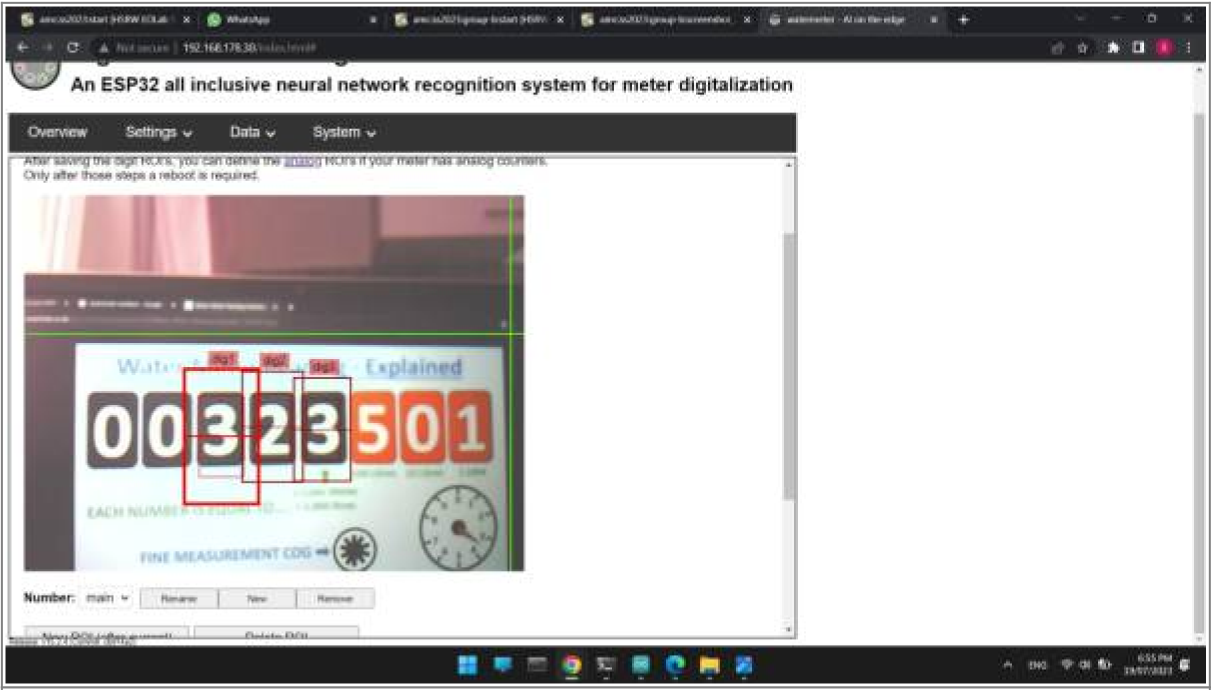
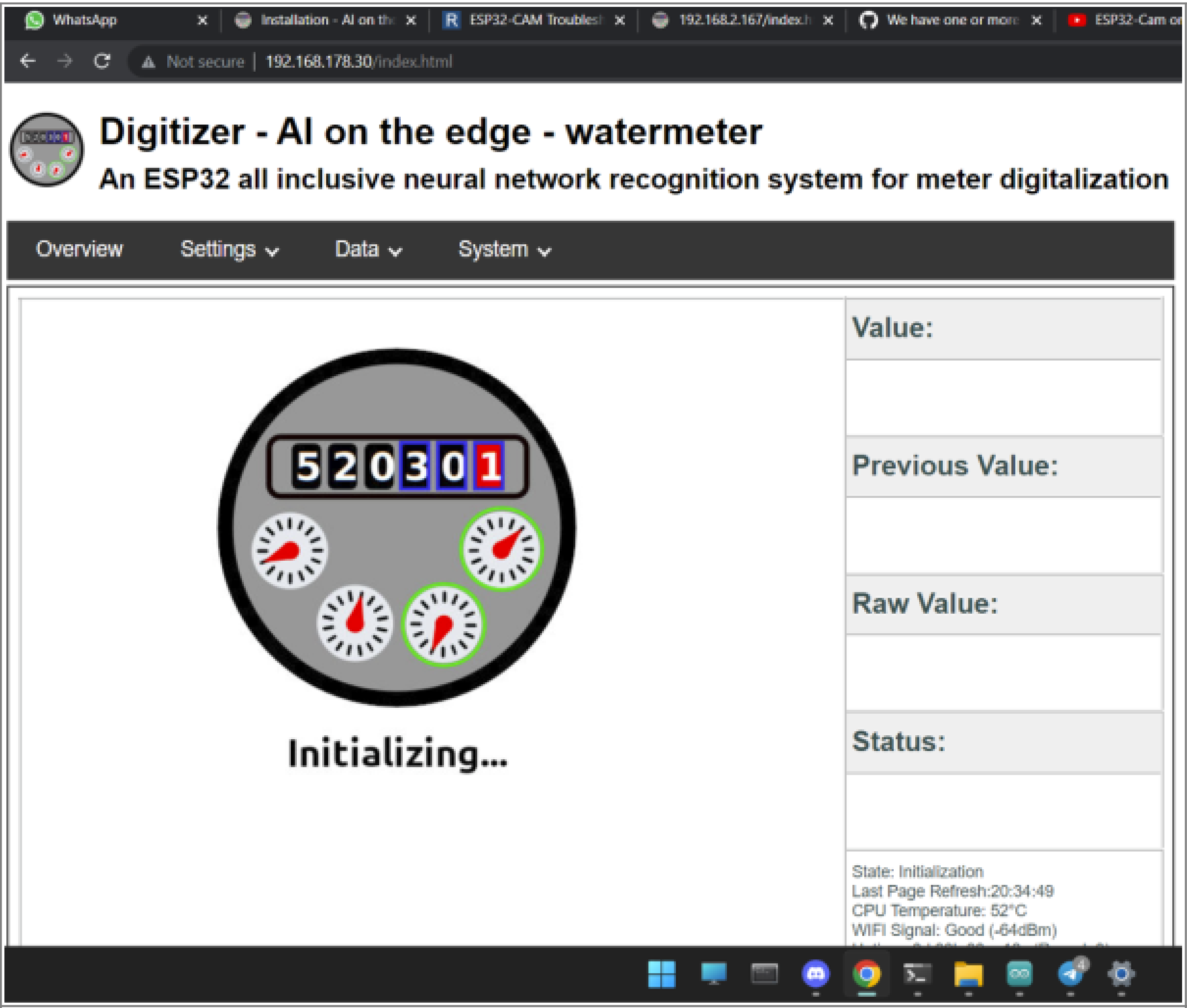

An ESP32 all inclusive neural network recognition system for meter digitalization
Overview
Settings
Data
System
After saving the digit ROIs, you can define the analog ROIs if your meter has analog counters.
Only after those steps a reboot is required.
Number: main
Rename
New
Remove

Fig.9: Reference image setting

The idea of the configuration is to align a proper box for the computer vision to work. By the box we mean that we have to indicate which field are suppose to be read. We can add or remove the boxes so finally we had to read the numbers and the analog indicator so just for test we made 3 number boxes and one analog indicator box. Then we can reboot the ESP32 module. The next step is to get the image of a water meter. We can check the status of the system in a serial monitor. There we can see that after rebooting and two failed attempts to connect to wifi we finally can start making photos (so-called “Rounds”). The round is a one time recognition.


Not secure | 192.168.178.30/index.html

Digitizer - AI on the edge - watermeter

An ESP32 all inclusive neural network recognition system for meter digitalization

Initializing...

Value:

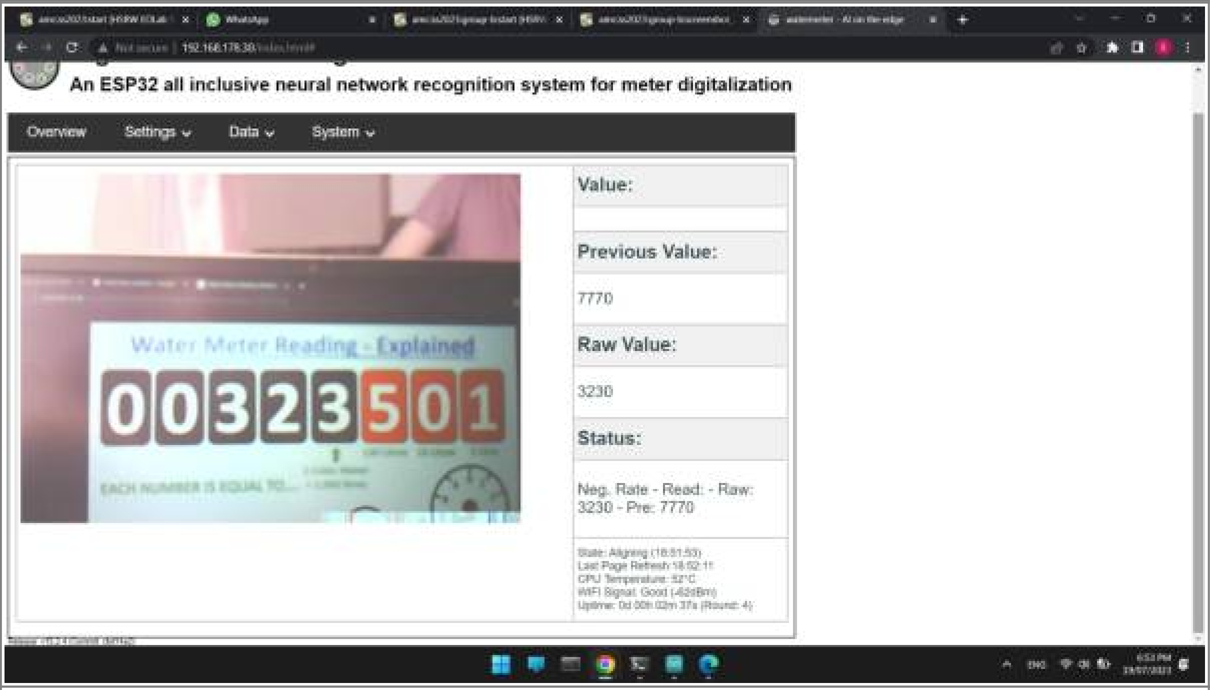
Previous Value:

Raw Value:

Status:

State: Initialization
Last Page Refresh:20:34:49
CPU Temperature: 52°C
WIFI Signal: Good (-64dBm)

Fig.10: Loading page


Not secure | 192.168.178.30/index.html

An ESP32 all inclusive neural network recognition system for meter digitalization

Value:

Previous Value:

7770

Raw Value:

3230

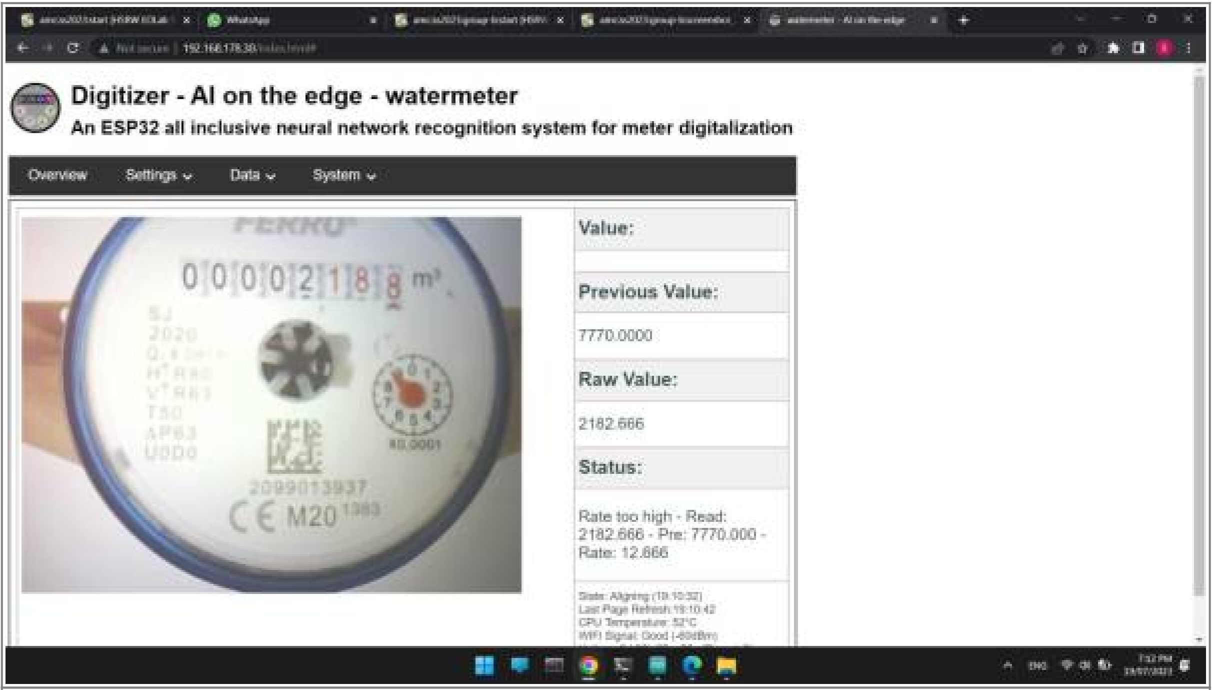
Status:

Neg. Rate - Read: - Raw: 3230 - Pre: 7770

State: Aligning (18:51:53)
Last Page Refresh 18:52:11
CPU Temperature: 52°C
WIFI Signal: Good (-62dBm)
Uptime: 0d 00h 02m 37s (Round: 4)

Fig.11: First reading

Initially, we had aligned our boxed with mistakes and we could clearly seen that the values are not correct. But after making proper boxes and making some improvements we finally got our recognised results. We then tried to recognise the analog values as well and we indeed got some results but we think that the system is not recognising the analog values properly. Also, it is clear that when the value is in between two states (between 8 and 9 for example) the system sometimes make mistakes. It should clearly see the value to get it so we think that it is useful to also check the photo of a water meter.



Digitizer - AI on the edge - watermeter

An ESP32 all inclusive neural network recognition system for meter digitalization

Value:

Previous Value:

7770.0000

Raw Value:

2182.666

Status:

Rate too high - Read: 2182.666 - Pre: 7770.000 - Rate: 12.666

State: Aligning (19:10:32)
Last Page Refresh: 19:10:42
CPU Temperature: 52°C
WIFI Signal: Good (-60dBm)

Fig.12: Final reading

Another setup

As it was previously said a number recognition process could make some mistakes in a lot of situations. Also, it could be not very convenient to open the website and check the results. But we have a solution for that situation.

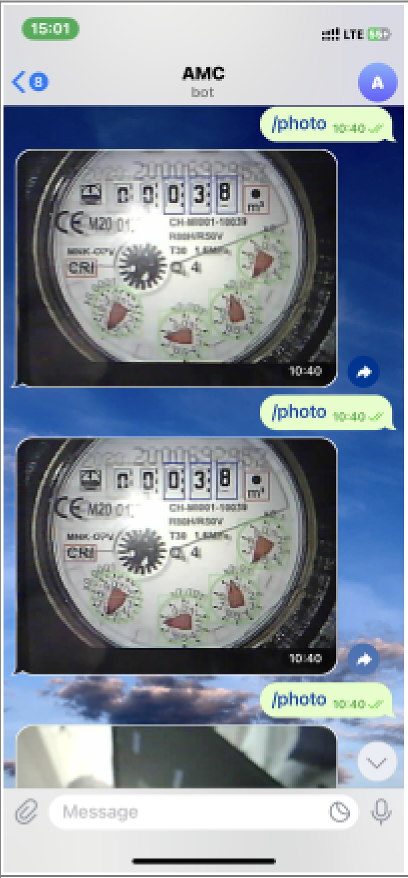


Fig.13: Telegram Bot Overview

We made a telegram bot which could be used for checking the values of the water meter. The process is very simple because we used a ready project that gives us a possibility to send commands on the ESP32 module and receive photos of a water meter.

We had to perform a couple of steps to make this bot usable:

1. Creating a new bot
2. Using the credentials from the created bot
3. Making the bot work
4. Receiving photos and check

After performing these operations we were able to get photos of the water meter directly onto our telegram account using /photos command and turning on the flash using /flash command.

For this project we used the code:

code.cs

```
#ifdef ESP32
  #include <WiFi.h>
#else
  #include <ESP8266WiFi.h>
#endif
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>   // Universal Telegram Bot Library
written by Brian Lough:
https://github.com/witnessmenow/Universal-Arduino-Telegram-Bot
```

```
#include <ArduinoJson.h>

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// Initialize Telegram BOT
#define BOTtoken "XXXXXXXXXX:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"  //
your Bot Token (Get from Botfather)

// Use @myidbot to find out the chat ID of an individual or a group
// Also note that you need to click "start" on a bot before it can
// message you
#define CHAT_ID "XXXXXXXXXX"

#ifdef ESP8266
  X509List cert(TELEGRAM_CERTIFICATE_ROOT);
#endif

WiFiClientSecure client;
UniversalTelegramBot bot(BOTtoken, client);

// Checks for new messages every 1 second.
int botRequestDelay = 1000;
unsigned long lastTimeBotRan;

const int ledPin = 2;
bool ledState = LOW;

// Handle what happens when you receive new messages
void handleNewMessages(int numNewMessages) {
  Serial.println("handleNewMessages");
  Serial.println(String(numNewMessages));

  for (int i=0; i<numNewMessages; i++) {
    // Chat id of the requester
    String chat_id = String(bot.messages[i].chat_id);
    if (chat_id != CHAT_ID){
      bot.sendMessage(chat_id, "Unauthorized user", "");
      continue;
    }

    // Print the received message
    String text = bot.messages[i].text;
    Serial.println(text);

    String from_name = bot.messages[i].from_name;

    if (text == "/start") {
      String welcome = "Welcome, " + from_name + ".\n";
      welcome += "Use the following commands to control your
outputs.\n\n";
      welcome += "/led_on to turn GPIO ON \n";
      welcome += "/led_off to turn GPIO OFF \n";
      welcome += "/state to request current GPIO state \n";
      bot.sendMessage(chat_id, welcome, "");
    }

    if (text == "/led_on") {
      bot.sendMessage(chat_id, "LED state set to ON", "");
      ledState = HIGH;
      digitalWrite(ledPin, ledState);
    }

    if (text == "/led_off") {
      bot.sendMessage(chat_id, "LED state set to OFF", "");
      ledState = LOW;
      digitalWrite(ledPin, ledState);
    }

    if (text == "/state") {
      if (digitalRead(ledPin)){
        bot.sendMessage(chat_id, "LED is ON", "");
      }
      else{
        bot.sendMessage(chat_id, "LED is OFF", "");
      }
    }
  }
}

void setup() {
  Serial.begin(115200);

  #ifdef ESP8266
    configTime(0, 0, "pool.ntp.org");      // get UTC time via NTP
    client.setTrustAnchors(&cert); // Add root certificate for
api.telegram.org
  #endif

  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, ledState);

  // Connect to Wi-Fi
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  #ifdef ESP32
    client.setCACert(TELEGRAM_CERTIFICATE_ROOT); // Add root
certificate for api.telegram.org
  #endif
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }
  // Print ESP32 Local IP Address
  Serial.println(WiFi.localIP());
}

void loop() {
  if (millis() > lastTimeBotRan + botRequestDelay)  {
    int numNewMessages = bot.getUpdates(bot.last_message_received + 1);

    while(numNewMessages) {
      Serial.println("got response");
      handleNewMessages(numNewMessages);
      numNewMessages = bot.getUpdates(bot.last_message_received + 1);
    }
    lastTimeBotRan = millis();
  }
}
```

Discussion

Despite these systems being effective in taking photos of the water meter we think that we have to improve the system of recognition. Without the system we cannot make a proper data analysis and a long time analysis so the usage of systems is limited for now.

It was a difficult task to make the camera work and connect it to a wifi while maintaining the connection between a camera and computer. We faced a lot of struggles:

1. Camera focus was very difficult to unblock from glue
2. Connection between camera and WiFi was always interrupted
3. Com port could not be found after a while so we had to restart the system and computer
4. The Ai-on-the-edge project website was not working
5. For telegram bot it was really hard to connect the number recognition system to a bot. There are some project on the internet, however, they use premade bots or paid ORS system. So we had to train the ai system by ourselves on a huge amount of data which was not possible due to a lack of data.

However, we made a big progress in maintaining the smart water meter system. A lot of improvements could be made but we proved that it is possible to extract data from the ESP32 module and use it to build smart solutions for easier life.

Links

https://randomnerdtutorials.com/telegram-esp32-cam-photo-arduino/

https://www.youtube.com/watch?v=iUgxwbfkIqU https://student-wiki.eolab.de/doku.php?id=start

From:
https://student-wiki.eolab.de/ - **HSRW EOLab Students Wiki**

Permanent link:
https://student-wiki.eolab.de/doku.php?id=amc:ss2023:group-b:start

Last update: **2023/07/24 21:23**