ESP32CAM liest Wasseruhr

Dank neuronaler Netze kann eine ESP32CAM die Ziffern und Zeiger analoger Wasseruhren auslesen und digitalisieren. Wir zeigen, wie man sowas nachbaut und einrichtet.

von Josef Müller

In diesem Do-It-Yourself-Projekt möchte ich euch eine robuste und einfach umsetzbare Möglichkeit zeigen, wie ihr den Zählerstand eures Wasserzählers an die Hausautomation und eure Datenerfassung anschließen könnt. Im Laufe der Jahre habe ich in meinem analogen Haus immer mehr Sensoren und Steuerungen eingesetzt und zum Beispiel aufgrund der Daten der Photovoltaikanlage die Rollläden wetterabhängig gesteuert.

Ein Messwert hatte sich jedoch in der Vergangenheit hartnäckig gegen seine Erfassung widersetzt: der Wasserverbrauch. Verschiedenste Versuche, unter anderem mit Reflexlichtschranken, waren nicht sehr zuverlässig oder langlebig. Erst ein neuer Ansatz über Bildverarbeitung, der quasi dem menschlichen Sehen nachempfunden ist, hat letztendlich bei meinem voll analogen Zähler zum Erfolg geführt.

In Bild 1 ist die Idee skizziert. Über eine leicht anpassbare Halterung nimmt eine Kamera ein Bild des Wasserzählers auf. Eine vorgegebene Referenzstruktur hilft beim Ausrichten der Kamera beziehungsweise des Bildes. Anschließend wird das Bild in die Ziffern und Zeiger zerlegt. Die Digitalisierung findet über integrierte neuronale Netze statt, die zuvor auf einem anderen System mit vielen Bildern von Ziffern und Zahlen trainiert wurden.

Da die neuronalen Netze sehr flexibel trainiert wurden, kann die Erkennung mit Zählern unterschiedlicher Bauart zurechtkommen und Ziffern und Zeiger trotz verschiedener Farben und Größen erkennen. Man muss also kein eigenes Netz mehr trainieren, sondern kann das fertige benutzen. Bild 2 zeigt einige Beispiele für Ziffern und Zeiger, die problemlos erkannt werden. Neue Ziffern und Zeiger lassen sich auch recht einfach ergänzen.

Kurzinfo

- **Neuronales Netz erkennt analoge Ziffern und Zeiger und wandelt Werte in digitale Daten um**
- **Montage der ESP32CAM auf der Wasseruhr**
- **Ausrichten und Kalibrieren der Kamerabilder**

Checkliste

Zeitaufwand:
2 Stunden ohne 3D-Druck

Kosten:
25 Euro

Programmieren:
Python mit esptool

Material

- **ESP32CAM**
- **FTDI-Adapter**
- **SD-Karte (max. 16GB)**
- **MicroUSB-Buchse**
- **Raspberry Pi 3 oder 4**
- **PLA-Filament**

Werkzeug

- **Lötkolben**
- **3D-Drucker**

Mehr zum Thema

- Daniel Bachfeld, Einstieg in KI, Make 6/18, S. 36
- Daniel Bachfeld, Intelligente Webcam für 5 Euro, Make 1/20, S. 28

Alles zum Artikel im Web unter
make-magazin.de/x9hb

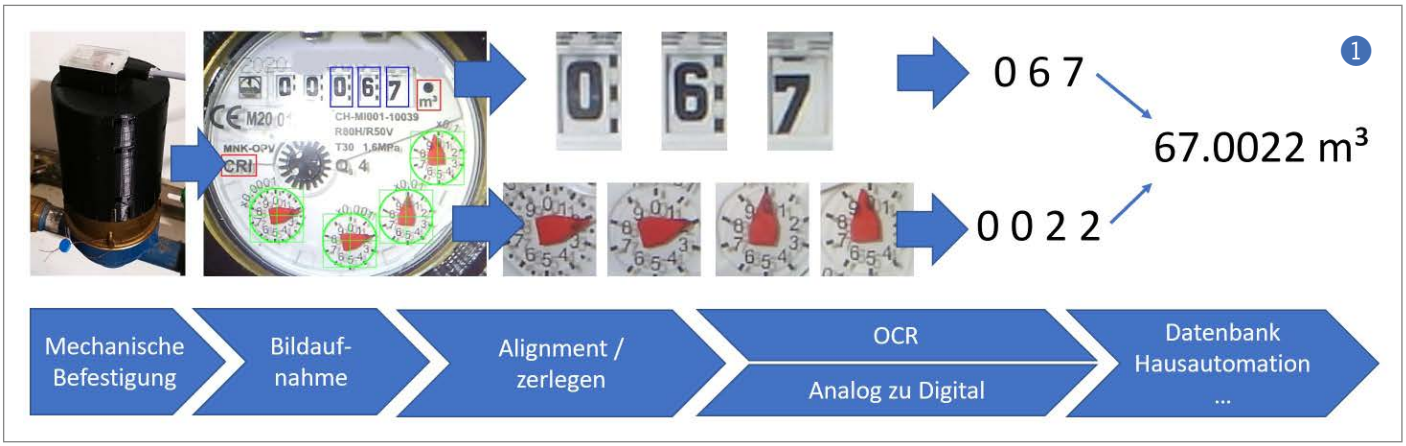
Systemüberblick

Die Idee zur jetzigen Version ist entstanden, als ich zum einen die günstige Kombination eines ESP32 mit einer Kamera in Form des ESP32CAM-Moduls und zum anderen die Ergänzung der Google Bibliothek *TensorFlow* um *TFlite* entdeckt habe. Diese Ergänzung macht das sogenante *Inferencing* mittels neuronaler Netze direkt auf Mikrocontrollern möglich.

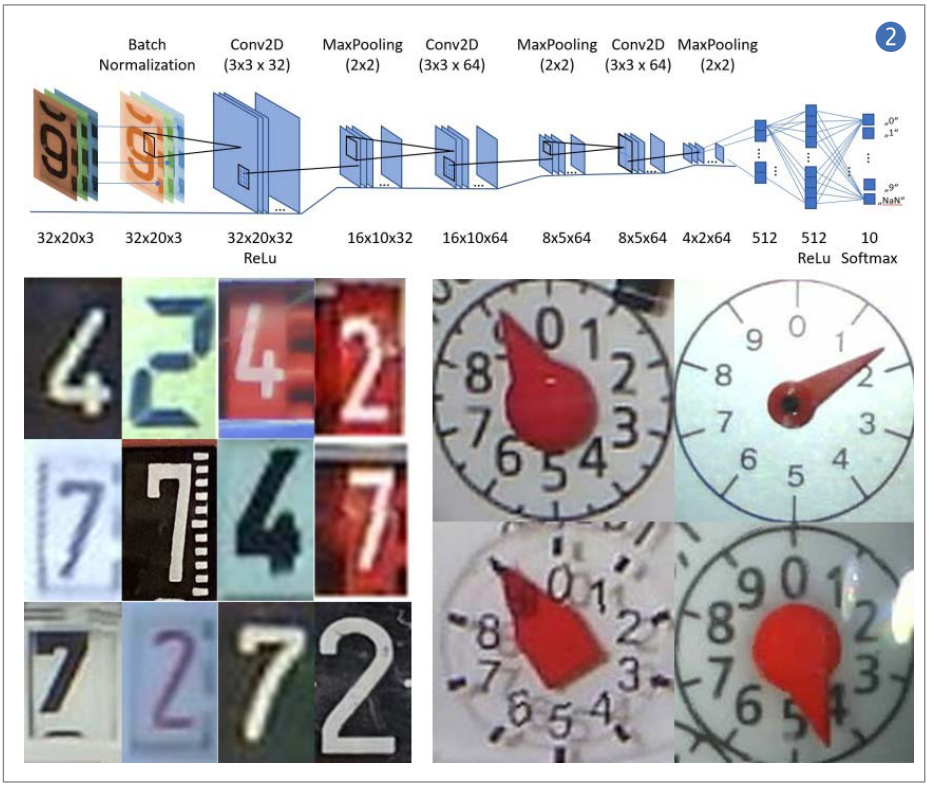
Das System besteht aus einer Hardware- und einer Softwarekomponente. Die Hardware dient vor allem dazu, ein reproduzierbares Bild aufzunehmen. Die Flexibilität der Zählererkennung liegt vor allem in der Software. Die Rechenpower des Zweikernprozessors vom Tensilica-Prozessor des ESP32 ermöglicht es, mehrere Prozesse parallel ablaufen zu lassen 3. Die Bildverarbeitung läuft immer wieder zyklisch ab und ermittelt laufend den aktuellen Zählerstand. Ein integrierter HTTP-Server erlaubt die Konfiguration des Systems per WLAN auch nach dem Einbau und ein MQTT-Client sendet den Zählerstand auf Wunsch an einen MQTT-Broker.

Ein bisschen Hintergrund für Interessierte: Anders als bei der vielen Lesern bereits bekannten Programmierung des ESP32 per Arduino-IDE habe ich meine Firmware mit der *ESP-IDF* erstellt, also der Entwicklungsumgebung des Herstellers *Espressif*. Das Betriebssystem ist *FreeRTOS*, das Multitasking mit Parallelprozessen ermöglicht. Doch keine Angst, ich stelle die bereits fertig übersetzte Firmware bereit, sodass man keine IDE installieren muss.

Die Bildverarbeitung im ESP32 ist verglichen mit Smartphones und PCs vergleichsweise langsam. Der Zählerstand lässt sich aber alle zwei bis vier Minuten ermitteln. Für die meisten Anwendungen sollte dies ausreichen.



1

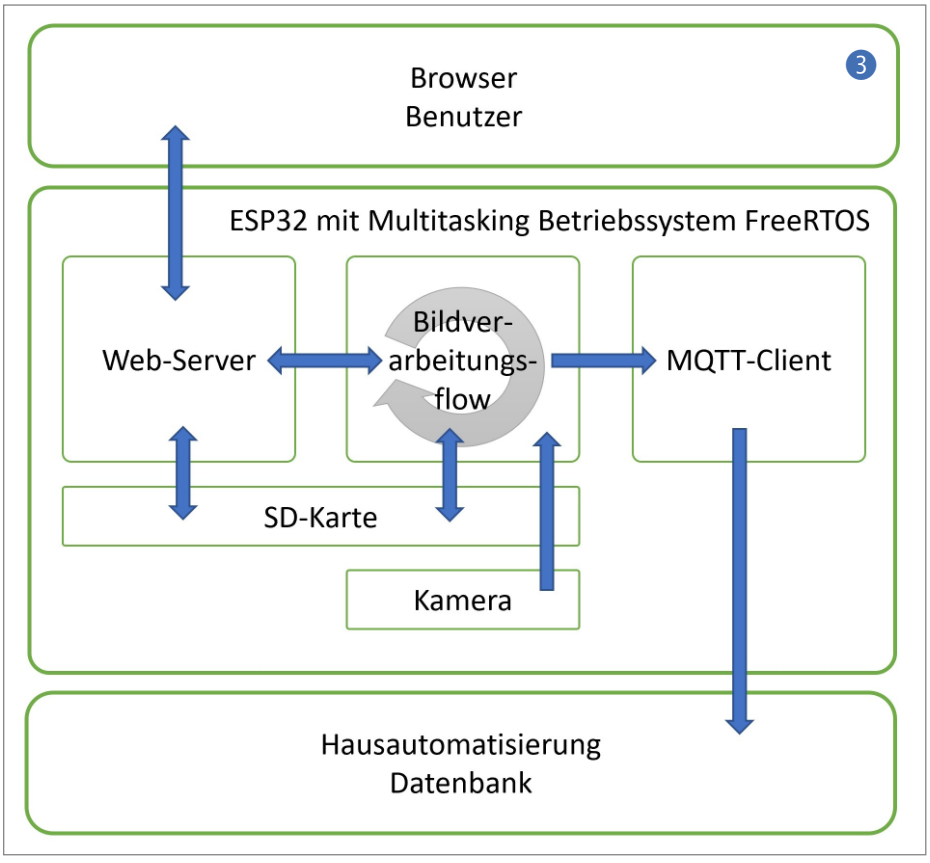


Die Details zur Bildverarbeitung und den neuronalen Netzen in TensorFlow würden den Rahmen dieses Artikels sprengen. Details finden sich auch auf meiner Projektseite im Internet (siehe Link in der Kurzinfo).

Hardware

Die Hardware ist in zwei Teile aufgeteilt: zum einen die Befestigung auf dem Wasserzähler und zum anderen ein Gehäuse für das ESP-32CAM-Modul inklusive Anschluss für die Stromversorgung (4). Für alle Komponenten gibt es STL-Dateien auf, sodass sie auf einem 3D-Drucker selbst hergestellt und auch an die eigenen Bedürfnisse angepasst werden können.

Idealerweise druckt man die Befestigung und den Boden der ESP32-Halterung in undurchsichtigem schwarzem Material. Dann kann später kein Umgebungslicht das Bild beeinflussen oder zusätzliche Reflexionen erzeugen. Den Deckel auf dem ESP32CAM-Modul habe ich transparent gedruckt. Dann kann man später noch die interne LED erkennen, die gewisse Statusänderungen, insbesondere beim Verbindungsaufbau, signalisiert (WLAN-Verbindung, Bildaufnahme). Die genaue Befestigung hängt natürlich vom Zähler ab. In den verschiedenen Foren zum Projekt gibt es mittlerweile sehr kreative Ansätze mit Hilfe von Chipsdosen, Plastikrohren oder schlicht einer Pappschachtel.



Halterung am Wasserzähler

Der Abstand zwischen Kamera und Oberfläche eines typischen Wasserzählers sollte circa 8cm betragen. Dann wird das Kamerabild durch den Wasserzähler vollflächig ausgefüllt. Die Halterung besteht aus Aufsatzring, Tubus (zweiteilig) und Deckel mit einer Aussparung und Clipshalterung für das ESP32CAM-Modul-Gehäuse. Der Aufsatzring wird einfach auf den Wasserzähler aufgesteckt. Falls dieser einen Deckel hat, muss man den eventuell aussparen oder abmontieren. An dieser Stelle kann man bei Bedarf das 3D-Model an die Gegebenheiten des Wasserzählers anpassen. Die beiden Tubusteile und der Deckel werden über einen Bajonettverschluss miteinander verbunden.

Egal, welche Art von Befestigung ihr wählt, wichtig ist hier vor allem, dass später ein reproduzierbares Bild aufgenommen wird und die Position der Kamera über dem Zähler (Höhe und Orientierung) unverändert bleibt.

ESP32-Halterung

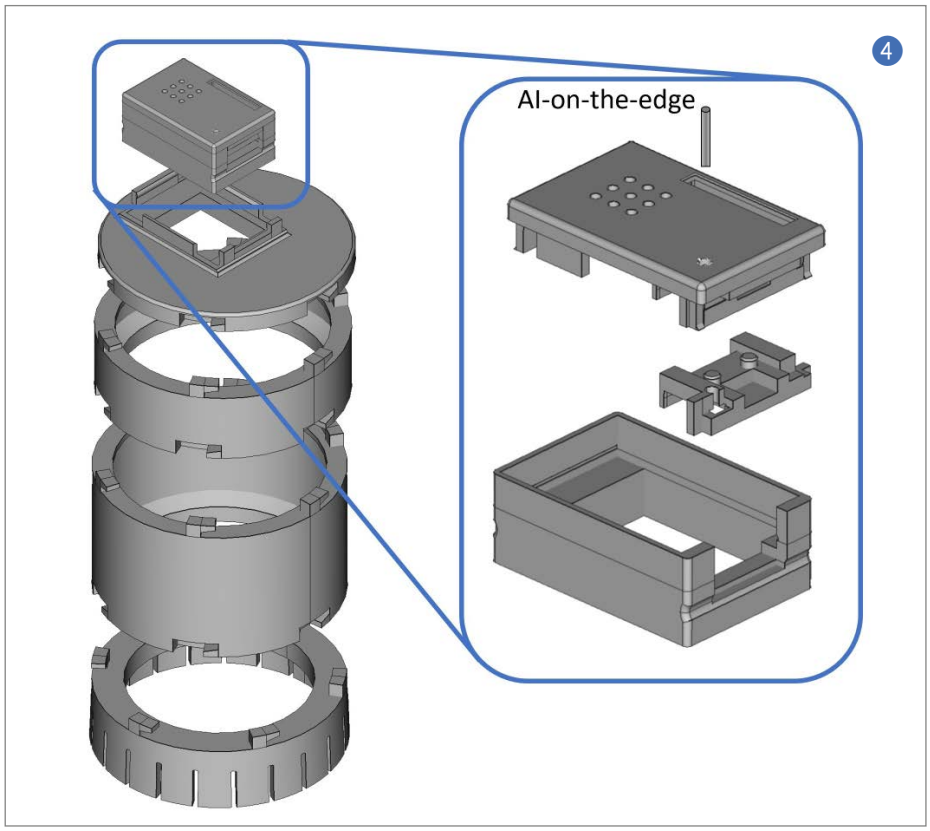
Das ESP32CAM-Modul ist in einer gesonderten Halterung untergebracht. Das Modul selbst kann man in Fernost für weniger als 10 Euro bekommen. Es besteht aus einer kompakten Kombination von ESP32, SD-Kartenleser, PSRAM, OV2640-Kamera und einer zusätzlichen Weißlicht-LED als integrierte Beleuchtung. Da das Modul jedoch keinen USB-Anschluss mitbringt, muss man die Stromversorgung über die entsprechenden PINs selbst sicherstellen. Ich rate zu einer Versorgung mit stabilen 5V.

Im Betrieb selbst benötigt man später nur noch die Stromversorgung. Alles andere wird

über die WLAN-Schnittstelle gesteuert. Auch spätere Firmware-Updates können über die Benutzeroberfläche eingespielt werden.

Lediglich die allererste Firmware muss über einen externen FTDI-Adapter manuell geflasht werden. Wie man den anschließt, zeigt Bild 5. Ich habe das so gelöst, dass ich über einen Micro-USB-Anschluss die Stromversorgung aus einen USB-Netzteil ziehe (GND, 5V). Das hat den Vorteil, dass ich zur Stromversorgung jedes beliebige USB-Netzteil verwenden kann. Auf die Pins, die zum Flashen notwendig sind, habe ich eine Stiftleiste aufgelötet. Das Gehäuse hat dort eine Aussparung, so dass man später auch von außen die Pins noch gut kontaktieren kann.

Die USB-Buchse sitzt auf einer Platine, die wiederum auf einem Plastikhalter befestigt und im Gehäuse platziert wird. Damit man auch den Reset-Taster erreicht, hat das Gehäuse dort ein Loch. Dort kann man einfach ein ca. 10mm lange Stück Filament (1,75mm Durchmesser) verwenden. Wenn man es an einem Ende abflacht, kann es auch nicht herausfallen. Im Gehäuse ist eine Aussparung für die Micro-SD-Karte frei gelassen, sodass man diese jederzeit entnehmen kann.



4

Kamerafokus

Die im ESP32CAM-Modul verbaute Kamera ist auf eine Tiefenschärfe von rund 40cm bis unendlich eingestellt. Da der Tubus mit 40cm sehr lang wäre und das Bild des Wasserzählers auf diese Entfernung auch sehr klein wird, passt man die Linse der Kamera besser an einen kurzen Abstand an 6. Das ist etwas diffizil und man verliert dabei vermutlich die Garantie – aus meiner Sicht aber bei dem geringen Preis verschmerzbar.

Die Schärfe wird über ein Feingewinde an der Objektivlinse eingestellt. Ein Tropfen Siegellack verhindert ein versehentliches Verdrehen. Dieser muss daher vorsichtig mit einem feinen Messer oder Schraubendreher gelöst werden. Dann kann die Linse mit einer kleinen Zange oder ähnlichem gedreht werden. Die eigentliche Justage erfolgt anhand des Kamerabilds nach der ersten Inbetriebnahme.

Firmware

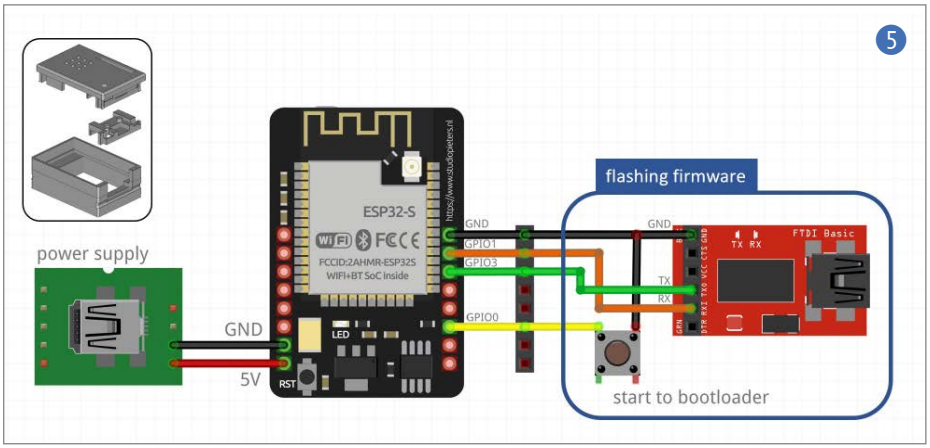
Die Firmware kann direkt aus Entwicklungsumgebungen wie *PlatformIO* oder *ESP-IDF* geflasht werden. Da die Konfiguration aber je nach System unterschiedlich ist, beschreibe ich nachfolgend das Flashen der fertig kompilierten Firmwaredateien über eine Python-Umgebung.

Das funktioniert bei unterschiedlichen Betriebssystemen sehr ähnlich (Windows mit Anaconda, Ubuntu, …). Hier zeige ich die Schritte am Beispiel eines Raspberry Pi. Auf diesem ist Python schon installiert und es sind nur noch wenige Schritte notwendig. Zum Schreiben der Firmware verwendet man am einfachsten das Python-Tool *esptool*. Dieses kann mit in einem Terminal in Raspberry leicht nachinstalliert werden (Befehl 1 in Kasten „Installation der Firmware“).

Die Firmware des ESP32 besteht aus drei Dateien: *bootloader.bin*, *partitions.bin* und *firmware.bin*. Der Bootloader und die Partitionstabelle müssen nur einmalig geflasht werden. Bei späteren Updates reicht es, nur die Firmware in den Flash-Speicher zu übertragen. Die Dateien müssen zunächst aus dem Github-Repository *AI-on-the-edge-device* heruntergeladen werden, am besten über den Befehl 2 (Kasten „Installation der Firmware“).

Ansonsten lädt man die Dateien als ZIP-Archiv von GitHub (siehe Link) und entpackt sie. Egal, welchen Weg man genommen hat, man sollte nun das Verzeichnis *AI-on-the-edge-device* sehen und mit dem Befehl `cd` dorthin wechseln.

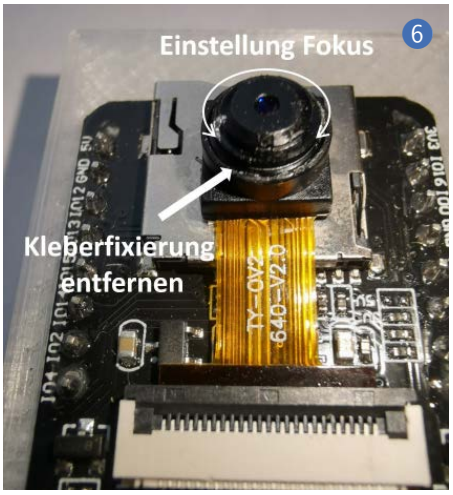
Vor dem Aufspielen der Firmware sollte sicherheitshalber der Speicher der ESP-32CAM komplett gelöscht werden. Dazu muss der ESP32 über die Verbindung von GPIO0 mit Masse (GND) während des Bootvorgangs in den Downloadmodus gebracht werden. Im Schaltbild ist dies durch den Taster realisiert. Anschließend initiiert man mit Befehl 4 den Löschvorgang – er dauert nur wenige Sekunden.



5

Installation der Firmware

```
1. sudo pip install esptool
2. git clone https://github.com/jomjol/AI-on-the-edge-device.git
3. cd AI-on-theedge-device/firmware
4. esptool erase_flash
5. esptool write_flash 0x01000 bootloader.bin 0x08000 partitions.bin 0x10000 firmware.bin
```



Jetzt wechselt man in das Verzeichnis *firmware*, in dem die drei Dateien liegen und gibt Befehl 5 ein. Die Firmware, der Bootloader und Partitionstabelle sind nun übertragen und geflasht und der ESP32 benötigt zum ersten Start nur noch die SD-Karte.

SD-Karte

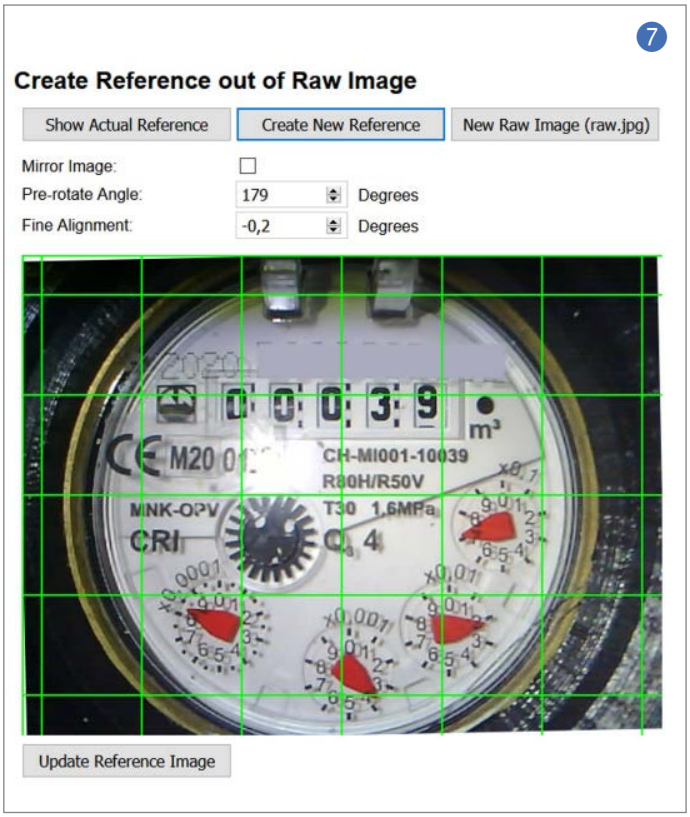
Die SD-Karte dient als Speicher für die Konfiguration, als Dateiablage für den Webserver, als Ablage für Logfiles und als Arbeitsverzeichnis für die Bildverarbeitung. Die Dateistruktur und die Grundkonfiguration muss vor dem ersten Start eingespielt werden. Dazu muss eine leere SD-Karte (max. 16GByte) mit dem Inhalt des Unterverzeichnisses sd-card bespielt werden.

Die WLAN-Zugangsdaten müssen in der Datei *wlan.ini* im Hauptverzeichnis der SD-Karte eingetragen werden. Dies ist die einzige Datei, die nachher nicht per HTTP-Fileserver zugänglich ist (rudimentärer Schutz des WLAN-Passwortes). Damit ist jetzt auch die Softwarekonfiguration abgeschlossen. Die SD-Karte wird eingesetzt und dem ersten Start steht nichts mehr im Weg.

Webinterface

Nachdem der ESP32 mit Strom versorgt wurde, startet er automatisch und wird sich zuerst mit dem WLAN verbinden. Die Startsequenz kann anhand der roten LED auf der Oberseite verfolgt werden. Nach einem kurzen Blinken wird eine erfolgreiche Verbindung durch dreimaliges, langsames Blinken angezeigt. Anschließend wird ein erstes Bild aufgenommen. Das wird durch ein 5 Sekunden langes Leuchten der LED während der Aufnahme angezeigt. Sollte die Verbindung nicht zustande kommen, sollte man neben den WLAN-Parametern auch die Signalstärke prüfen, die manchmal in Kellern sehr schwach sein kann.

Über den Router oder die serielle Schnittstelle (Monitor am FTDI-Anschluss) findet man die IP-Adresse und schon kann man über einen Webbrowser direkt darauf zugreifen. Standardmäßig wird als Hostname *watermeter* gesetzt, sodass der ESP32 auch über *http://*



watermeter erreichbar sollte. Per Default-Konfiguration geht der ESP32 in einen initialen Setup-Modus, wenn das Webinterface das erste Mal aufgerufen wird.

Konfiguration

Im initialen Setup-Modus wird man durch die notwendigen Schritte zum Einrichten der Kamera geleitet. Damit die Konfiguration leichter fällt, gebe ich zunächst einen kurzen Überblick über die Idee hinter dem Ablauf, bevor Detailinformationen zu den einzelnen Schritten folgen.

Grundsätzlich könnte man auf einem aufgenommenen Kamerabild die Zeiger und Zähler markieren und anschließend kann der Erkennungsalgorithmus das Bild entsprechend zerlegen und die Erkennung durchführen. Genau das passiert auch, aber erst in einem zweiten Schritt. Denn der Nachteil dieses sehr einfachen Verfahrens ist es, dass es sehr empfindlich gegenüber kleinsten Verschiebungen der Kameraposition ist, zum Beispiel wenn man die Halterung für eine Kontrolle abnehmen muss oder man mal am Kabel wackelt. Das wäre sehr anfällig und mit einem hohen Wartungsaufwand (häufige Korrektur der Konfiguration) verbunden.

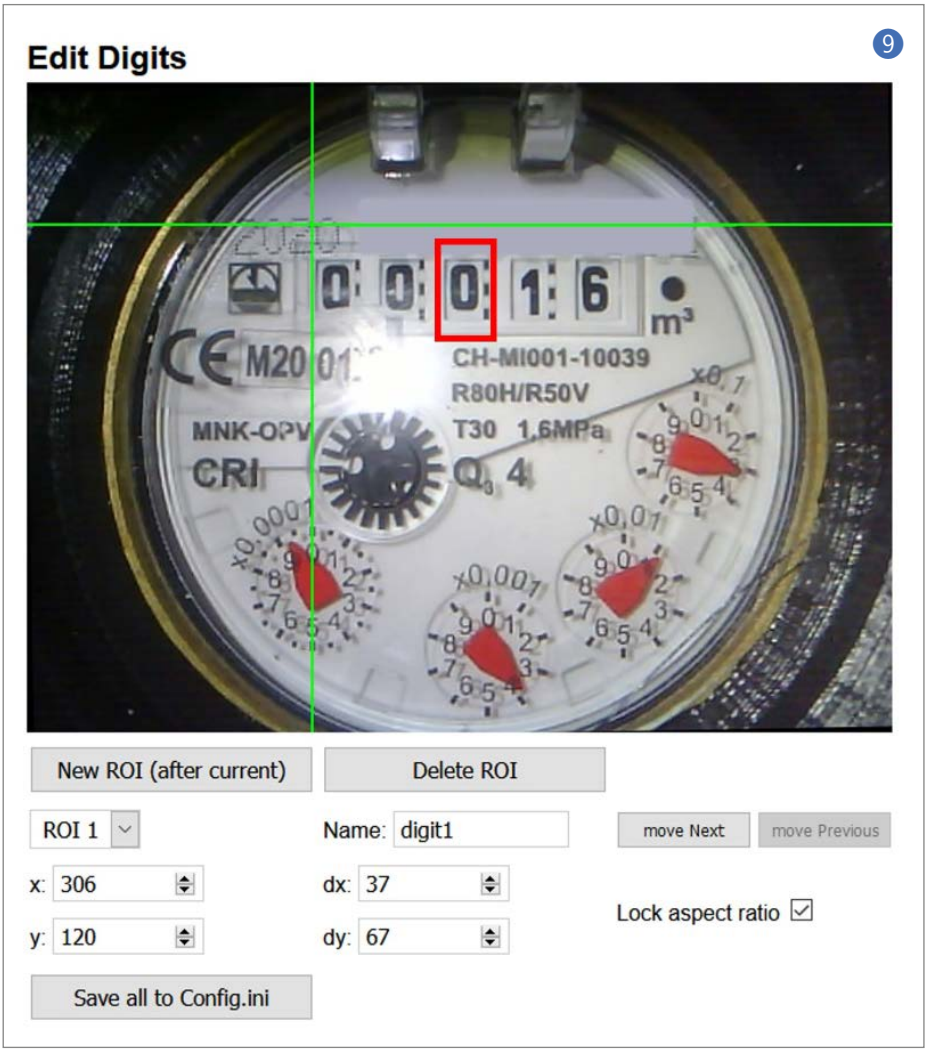
Daher ist vor dem Ausschneiden der Ziffern und Zähler noch ein Algorithmus zum Ausrichten des Kamerabildes vorgeschaltet. Dieser kann typische Verschiebungen und Verdrehungen, wie sie beim Abnehmen und wieder Aufsetzen der Halterung vorkommen, automatisch korrigieren, ohne dass jedes Mal die Positionen der Ziffern und Zeiger neu markiert werden müssen.

Dieser Algorithmus basiert auf dem Prinzip eines Referenzbildes mit eindeutigen Positionsreferenzen. Diese Referenzen werden später auf dem frisch aufgenommenen Kamerabild gesucht und damit die Koordinaten umgerechnet. Das resultiert in einem sehr stabilen Gesamtalgorithmus.

Die Konfiguration besteht aus fünf Schritten, die im Nachfolgenden erklärt werden. Nachdem man diese durchlaufen hat, wird der ESP32 neu gestartet und erkennt fortan den Zählerstand.

Referenzbild erstellen

Wie oben beschrieben, dient das Referenzbild zur Bildausrichtung und Markierung der Ziffern und Zeiger. Dies ist auch der erste Schritt im Setup. Ihm kommt eine besondere Bedeutung zu und man sollte diesen Schritt sehr gewissenhaft durchführen. Ziel ist es, ein scharfes und optimal ausgerichtetes Bild zu bekommen. Letzteres bedeutet, dass die Ziffern und Zeiger nicht verdreht sind, sondern exakt ausgerichtet sind. Dazu werden auch Hilfslinien im Bild eingeblendet (7).



Zunächst wird noch das auf der SD-Karte gespeicherte Dummy-Referenzbild der Default-Konfiguration angezeigt. Das Erstellen einer angepassten Referenz wird durch den Button *Create New Reference* gestartet. Jetzt wird das neu aufgenommene Kamerabild angezeigt und man kann über die Winkeleinstellungen (Grob- und Feinjustage) eine exakte Ausrichtung erreichen.

Wenn das Bild noch nicht perfekt scharf ist, sollte man das Objektiv jetzt justieren. Dazu muss man die Halterung entfernen und die Objektivlinse drehen (typischerweise reindrehen) und kann nach dem Aufsetzen der Halterung über *Take Image* das Kamerabild aktualisieren und die Bildschärfe kontrollieren.

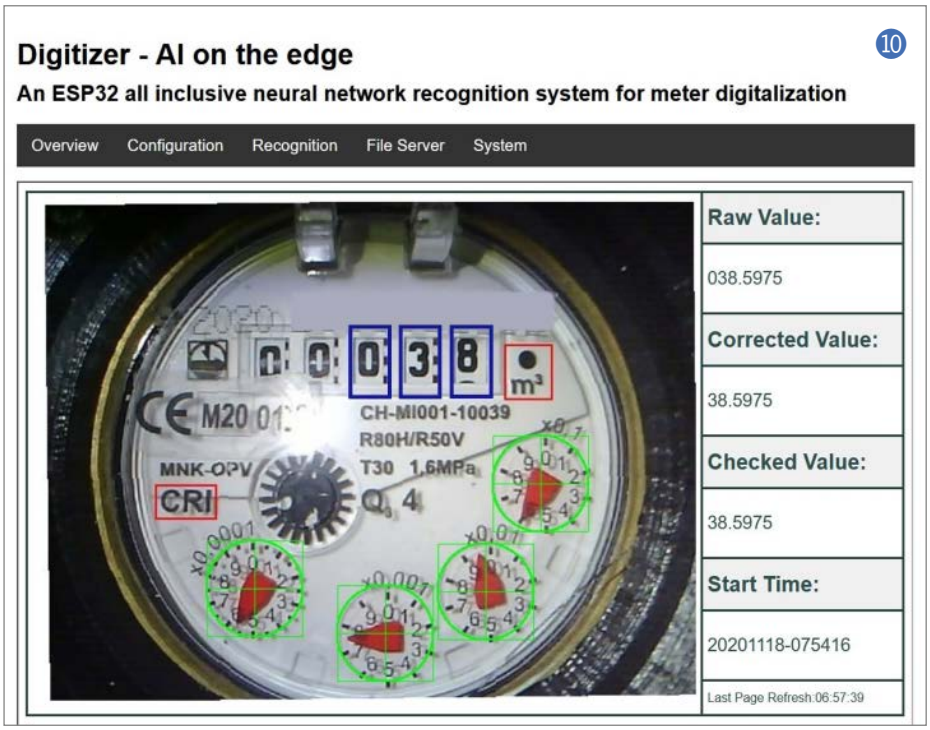
Ein wichtiger Aspekt für die Ausrichtung sind Reflexionen. Je nach Wasserzähler können diese ungünstig über einer Ziffer oder einem Zeiger liegen. Dann hilft es, wenn man die Halterung verdreht, sodass die Reflexion an einer unkritischen Stelle ist. Die Drehung kann man dann über die Softwareeinstellungen wieder korrigieren. Manchmal hilft es auch, an der Stelle der Reflexion ein Stück schwarzen Stoff oder Pappe auf das Zählerglas zu kleben.

Erst wenn Schärfe und Position der Halterung final festgelegt sind und das Bild gerade ausgerichtet ist, sollte man die Einstellungen über *Update Reference Image* speichern. Dieses Referenzbild ist dann die Basis für die folgenden Einstellungen.

Bildreferenzen definieren

Im nächsten Schritt wird definiert, anhand welcher Strukturen später das einzelne aufgenommene Bild ausgerichtet wird. Für die Korrektur von Versatz und Verdrehung sind zwei Referenzen notwendig. Es ist wichtig, hier nicht zu große, aber doch eindeutige Strukturen auszuwählen. Die Referenzen sollten möglichst gegenüber liegen. Bei meinem Zähler habe ich gute Erfahrungen mit der Schrift m^3 und dem Schriftzug *CRI* gemacht.

Man sieht in diesem Schritt das eben erzeugte Referenzbild und kann entweder über die Koordinaten im Textfeld oder über die Maus (Ecke markieren, Maustaste gedrückt halten und ziehen) die Referenz markieren (8).

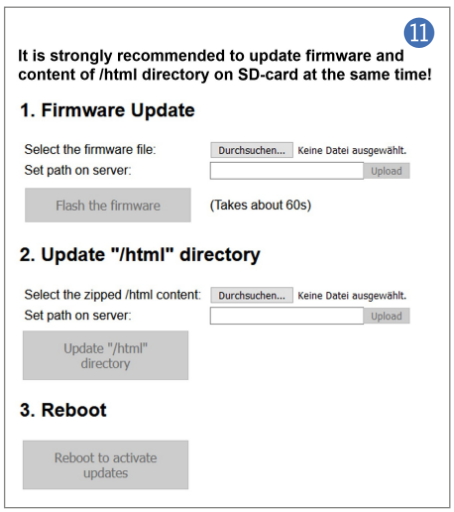


10

Erst mit dem Button *Update Reference* wird der Bildausschnitt ausgewählt und das Referenzbild wird aktualisiert. Zwischen den beiden Referenzen kann über das Drop-Down Menü gewechselt werden. Erst durch den Schalter *Save to Config.ini* werden die Änderungen für beide Referenzen in die Konfiguration übernommen. Jetzt fehlt eigentlich nur noch die Markierung der Ziffern und Zeiger.

Markierung von Ziffern und Zeiger

Die Definition der Ziffern und Zeiger erfolgt auch über eine grafische Benutzeroberfläche sehr ähnlich zu den Referenzen. Zunächst werden im Setup-Menü die Ziffern und im nächsten Schritt die Zeiger markiert. Beides ist sehr ähnlich, sodass ich es hier am Beispiel der Ziffern erkläre.



11

Man sieht auch in diesem Schritt zunächst das Referenzbild und die Markierung für die erste Ziffer (auch *Region-Of-Interest* = ROI genannt) 9. Die zu bearbeitende Ziffer wird auch hier über ein Drop-Down-Menü ausgewählt und kann dann genauso wie schon die Referenzen mittels Maus oder direkter Eingabe der Koordinaten bearbeitet werden. Im Unterschied zu vorher können nun jedoch weitere ROIs erzeugt oder auch wieder gelöscht werden, um die Erkennung an die Anzahl der Ziffern bzw. Zeiger anzupassen.

Hier spielt die Reihenfolge eine wichtige Rolle. Sie definiert die Ordnung, in denen anschließend die einzelnen Ziffern zum Zählerstand zusammengesetzt wird. Daher kann sie über *move Next* / *move Previous* verändert werden. Auch hier gilt: erst durch *Save all to Config.ini* werden die Änderungen übernommen und abgespeichert.

Allgemeine Konfigurationsparameter

Im letzten Schritt können nun noch weitere Konfigurationsparameter eingestellt werden. Eine Tabelle im Github-Wiki führt alle Parameter mit den Werten und jeweils einer kurzen Erklärung auf. Es kann zwischen normalem und Expertenmodus gewechselt werden. Für den ersten Versuch würde ich jedoch empfehlen, die Grundeinstellungen erstmal zu übernehmen. Im Wesentlichen sind die typischen Korrekturalgorithmen sinnvoll eingestellt.

Falls man den Zählerstand automatisch an einen MQTT-Broker übertragen will, kann man hier jetzt unter dem dementsprechenden Abschnitt die notwendigen Einträge vornehmen. Alle Einstellungen sind nachher auch über die ganz normalen Menüs zugänglich und lassen sich dort jederzeit anpassen. Zu beachten ist, dass ein Update der Parameter immer einen Neustart erfordert, damit die Änderungen auch korrekt übernommen werden. Dies wird auf der letzten Seite des Setups initiiert. Dort wird gleichzeitig auch noch der initiale Setup-Modus deaktiviert. Nach dem Reboot gelangt man jetzt direkt auf die Ergebnisseite, wo das aktuelle Bild und die ausgelesenen Werte angezeigt werden. Eventuell ist ein manueller Reload der Seite nach 10 bis 20 Sekunden notwendig. Das Auslesen des Zählerstandes startet nun automatisch und dauert ca. zwei bis vier Minuten. Solange muss man also Geduld haben, bis bei *Raw Value* ein sinnvoller Wert erscheint 10.

Ziel erreicht

Damit kann man jetzt den Zählerstand jederzeit abfragen. Neben der Anzeige auf der zentralen HTTP-Seite gibt es vor allem zwei Möglichkeiten, die Daten abzufragen: Zum einen können die aktuellen Ergebnisse über eine http-URL abgefragt werden und dann über einen Parser weiterverarbeitet werden. Die URL lautet:

http://IP-ADRESSE/wasserzaehler.html

Sowohl die Firmware wie auch die HTML-Seiten sind nach wie vor in der Weiterentwicklung auf Github durch eine aktive Community. Neben der Fehlerkontrolle wird der Algorithmus auch stetig verbessert und erweitert. Damit man bei einem Update nicht jedes Mal den ESP32 ausbauen muss, ist ein Over-The-Air-Update (OTA-Update) sowohl für die Firmware wie auch für den HTML-Server implementiert. Die neuen Dateien findet man auf Github immer im Verzeichnis *firmware* auf der Projektseite. Sie können über den Menüpunkt *System OTA Update* auf die ESP32CAM hochgeladen und installiert werden 11.

Im Prinzip reicht die *Datei firmware.bin*, für ein Update der HTML-Seiten lädt man *html.zip* hoch. Letztere enthält eine gezippte Version des Verzeichnisses */html*. Nach dem Upload kann man die Firmware flashen bzw. die Dateien des HTTP-Servers austauschen. Im Zweifelsfall sollten immer beide Dateien ausgetauscht werden, da die Funktionen voneinander abhängig sind. Das Hochladen und auch der Austausch können einige Zeit dauern. Damit dann die Änderungen aktiv sind, ist noch ein Reboot notwendig. Unter *System Info* kann man die aktuelle Version ablesen.

So – und nun viel Spaß beim Nachbauen und Ablesen des Wasserzählerstandes oder eines anderen Zählers, der aus Ziffern und/oder Zeigern besteht! —*dab*