IoT-Projekt (Eingangskontrolle)

für IoT-Projekt 2026

- Camera Erkennung
  - Links für Raspberry Pi einrichtung
  - Verläufiger Code für die Kameraerkennung mit mqtt übertragung
  - Stabiler Code vor mqtt ergänzung
  - Grundlagen + Infos

- LED Konfiguration
- MQTT-Broker Kram
- Manuelles ESP-32 Flashen mit Tasmota über Windows Terminal
- RFID Konfiguration
- Credentials
- Doku und Präsi
- Bilder

Camera Erkennung

Camera Erkennung

Links für Raspberry Pi einrichtung

Benutztes Modell: Yolo 26n

Installationsanleitung

Benötigte Applications:

Pytorch

Ultralytics

Python Version 3.8 oder Älter

Verläufiger Code für die Kameraerkennung mit mqtt übertragung

```
Code erstellt/angepasst mit Perplexity/Claude

import cv2
import ssl
import math
import paho.mqtt.client as mqtt
from ultralytics import YOLO

# ------------------ Einstellungen ------------------

MODEL_PATH = r"C:\Cam\yolo26n.pt"
SOURCE = 1

LINE_P1 = (300, 400)
LINE_P2 = (200, 100)
LINE_BUFFER = 30

MIN_MOVE_PIXELS = 5

# ------------------ Logik-Variablen ------------------

Room_count = 0
```

```python
last_positions = {}

person_states = {}

crossed_ids = set()



# ----------------- Hilfsfunktionen -----------------


def point_side_of_line(px, py, x1, y1, x2, y2):

    return (x2 - x1) * (py - y1) - (y2 - y1) * (px - x1)



def get_zone(px, py):

    val = point_side_of_line(px, py, LINE_P1[0], LINE_P1[1], LINE_P2[0], LINE_P2[1
])

    length = math.sqrt((LINE_P2[0]-LINE_P1[0])**2 + (LINE_P2[1]-LINE_P1[1])**2)

    dist = abs(val) / length

    if dist < LINE_BUFFER:

        return "zone"

    return "right" if val > 0 else "left"



def draw_info(frame):

    cv2.line(frame, LINE_P1, LINE_P2, (0, 255, 255), 2)

    cv2.putText(frame, f"Raum: {Room_count}", (10, 30),

                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)



# ----------------- Hauptprogramm -----------------


def main():

    global Room_count, last_positions, person_states, crossed_ids


    model = YOLO(MODEL_PATH)
```

```python
    mqtt_client = mqtt.Client(callback_api_version=mqtt.CallbackAPIVersion.VERSION2
)

    mqtt_client.username_pw_set("testuser", "O#%~54{Kf{-c-7t")

    mqtt_client.tls_set(tls_version=ssl.PROTOCOL_TLS_CLIENT)

    mqtt_client.connect("5ea1e51a4f614745a394b1edd0259a6d.s1.eu.hivemq.cloud", 8883
, 60)

    mqtt_client.loop_start()

    print("Starte Kamera... Drücke 'q' zum Beenden")


    for result in model.track(

        source=SOURCE,

        show=False,

        stream=True,

        persist=True,

        classes=[0],

        conf=0.5,         # ? reduzierter Schwellwert

        verbose=False,

    ):

        frame = result.plot()

        draw_info(frame)


        if result.boxes is not None and len(result.boxes) > 0:

            boxes = result.boxes

            track_ids = boxes.id


            if track_ids is not None:

                active_ids = set()


                for box, tid in zip(boxes.xyxy, track_ids):

                    track_id = int(tid.item())

                    active_ids.add(track_id)
```

```
                    x1, y1, x2, y2 = box.tolist()
                    cx = int((x1 + x2) / 2)
                    cy = int((y1 + y2) / 2)

                    cv2.circle(frame, (cx, cy), 4, (255, 0, 0), -1)
                    cv2.putText(frame, f"ID {track_id}", (cx + 5, cy - 5),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)

                    current_zone = get_zone(cx, cy)

                    if track_id not in person_states:
                        if current_zone != "zone":
                            person_states[track_id] = current_zone
                    else:
                        prev_zone = person_states[track_id]

                        if (prev_zone == "left" and current_zone == "right") or \
                           (prev_zone == "right" and current_zone == "left"):

                            if track_id not in crossed_ids:
                                crossed_ids.add(track_id)

                                if current_zone == "right":
                                    Room_count -= 1
                                    if Room_count < 0:
                                        Room_count = 0
                                    print(f"ID {track_id} EXITED, Raum: {Room_count
}")
                                    mqtt_client.publish("raum/personen",
                                                        f"ID{track_id}
,event=EXIT,raum={Room_count}")
```

```
                                else:
                                    Room_count += 1
                                    print(f"ID {track_id} ENTERED, Raum: {
Room_count}")
                                    mqtt_client.publish("raum/personen",
                                                        f"ID{track_id}
,event=ENTER,raum={Room_count}")

                    if current_zone != "zone":
                        person_states[track_id] = current_zone

                last_positions[track_id] = (cx, cy)

            gone_ids = crossed_ids - active_ids
            crossed_ids -= gone_ids
            for gid in (set(person_states.keys()) | set(last_positions.keys
())) - active_ids:
                person_states.pop(gid, None)
                last_positions.pop(gid, None)

        cv2.imshow("YOLO Room Counter", frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cv2.destroyAllWindows()
    mqtt_client.loop_stop()
    mqtt_client.disconnect()
    print(f"\nFinale Zählerstände: Raum={Room_count}")


if __name__ == "__main__":
    main()
```

Stabiler Code vor mqtt ergänzung

```python
import cv2
from ultralytics import YOLO


# ------------------ Einstellungen ------------------


MODEL_PATH = r"C:\Cam\yolo26n.pt"   # Pfad zu deinem .pt
SOURCE = 0                          # 0 = Standard-Webcam


# Linie (Türschwelle) im Bild anpassen!
LINE_P1 = (300, 400)   # Punkt 1 (x1, y1)
LINE_P2 = (200, 100)   # Punkt 2 (x2, y2)


# Minimale Bewegung, damit ein Seitenwechsel gezählt wird
MIN_MOVE_PIXELS = 5


# ------------------ Logik-Variablen ------------------


Room_count = 0
last_positions = {}  # {track_id: (x, y)}


# ------------------ Hilfsfunktionen ------------------


def point_side_of_line(px, py, x1, y1, x2, y2):
    """
```

```
    Gibt das Vorzeichen der Punktlage relativ zur Linie zurück.
    >0: eine Seite, <0: andere Seite, 0: genau auf der Linie.
    """
    return (x2 - x1) * (py - y1) - (y2 - y1) * (px - x1)


def draw_info(frame):
    """Zähler und Linie ins Bild zeichnen."""
    cv2.line(frame, LINE_P1, LINE_P2, (0, 255, 255), 2)
    cv2.putText(frame, f"IN: {Room_count}", (10, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)


# ------------------ Hauptprogramm ------------------

def main():
    global Room_count, last_positions

    # Modell laden
    model = YOLO(MODEL_PATH)

    print("Starte Kamera... Drücke 'q' zum Beenden")

    # Tracking-Loop
    for result in model.track(
        source=SOURCE,
        show=False,
        stream=True,
        persist=True,
        classes=[0],       # nur "person"
        verbose=False,
```

```
    ):
        # Bild holen (mit eingezeichneten Boxen)
        frame = result.plot()
        draw_info(frame)

        # Boxen / Tracks auswerten
        if result.boxes is not None and len(result.boxes) > 0:
            boxes = result.boxes
            track_ids = boxes.id

            # WICHTIG: track_ids kann None sein, wenn keine Tracks vorhanden
            if track_ids is not None:
                for box, tid in zip(boxes.xyxy, track_ids):
                    track_id = int(tid.item())
                    x1, y1, x2, y2 = box.tolist()
                    cx = int((x1 + x2) / 2)
                    cy = int((y1 + y2) / 2)

                    # Schwerpunkt einzeichnen
                    cv2.circle(frame, (cx, cy), 4, (255, 0, 0), -1)
                    cv2.putText(frame, f"ID {track_id}", (cx + 5, cy - 5),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)

                    # Vorherige Position holen
                    if track_id in last_positions:
                        prev_cx, prev_cy = last_positions[track_id]

                        # Nur auswerten, wenn sich die Person genug bewegt hat
                        dist_sq = (cx - prev_cx) ** 2 + (cy - prev_cy) ** 2
                        if dist_sq >= MIN_MOVE_PIXELS ** 2:
                            # Seite relativ zur Linie vorher / nachher
```

```
                        prev_side = point_side_of_line(prev_cx, prev_cy,
                                                       LINE_P1[0], LINE_P1[1],
                                                       LINE_P2[0], LINE_P2[1])
                        curr_side = point_side_of_line(cx, cy,
                                                       LINE_P1[0], LINE_P1[1],
                                                       LINE_P2[0], LINE_P2[1])

                        # Seitenwechsel -> Linie überquert
                        if prev_side * curr_side < 0:
                            # Richtung über y-Bewegung (an Kamera anpassen!)
                            if cx > prev_cx:
                                Room_count -= 1
                                if Room_count < 0:
                                    Room_count = 0
# Verhindert negativen Zähler
                                print(f"ID {track_id} ENTERED, IN: {Room_count}
")
                            else:
                                Room_count += 1
                                print(f"ID {track_id} EXITED, IN: {Room_count}"
)

                    # aktuelle Position speichern
                    last_positions[track_id] = (cx, cy)

        # Bild anzeigen
        cv2.imshow("YOLO Room Counter", frame)

        # Nur mit 'q' beenden
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
```

```
    cv2.destroyAllWindows()
    print(f"\nFinale Zählerstände: IN={Room_count}")


if __name__ == "__main__":
    main()
```

Grundlagen + Infos

Wir verwenden für dieses Projekt das bereits trainierte Personenerkennungsmodul von Ultralytics "YOLO"
Hierbei wird die neuste Version "yolo26n" benutzt.

Ultralytics hat hierfür bereits eine Python Library erstellt mit allen wichtigen Variablen und Funktionen bereits vordefiniert.

Das Personenerkennungsmodul ist als .pt (python torch) abgespeichert.

(

```
import cv2
import ssl
import paho.mqtt.client as mqtt
from ultralytics import YOLO
)
```

LED Konfiguration

=> Im Anhang liegen Dokumente für die Konfiguration.

WICHTIG: Voraussetzungen sind eine Tasmota geflashte ESP-32 und zusätzlich das ausgewählte Modul "**WS2812**" auf einen der Data-Pins der ESP-32! (In unserem Fall GPIO04).

ESP-32 Kabelverknüpfung:
Gehen wir davon aus, dass wir von oben auf die ESP-32 schauen und der USB-C Port nach unten schaut...

GND: 1. Reihe, außen (auf dem rechten Pinbrett)
Data: 6. Reihe, außen (auf dem rechten Pinbrett)
Voltage: 8. Reihe innen (auf dem linken Pinbrett)

Picture:

Licht AN/AUS
Topic: cmnd/esp32-led/POWER
Payload/Message: off **ODER** on

Licht TOGGLEN
Topic: cmnd/esp32-led/POWER
Payload/Message: toggle

Licht steuern
Topic: cmnd/esp32-led/Led
Payload/Message: <RGB/Hex> <RGB/Hex> etc...
=> Beispiel Payload/Message: 255,0,0 0,255,0 0,255,0
(Ersten drei Lichter werden gesteuert)

Um einzelne Lichter auszuschalten, setze Werte auf 0!
=> Beispiel Payload/Message: 0,0,0 **ODER** 000000

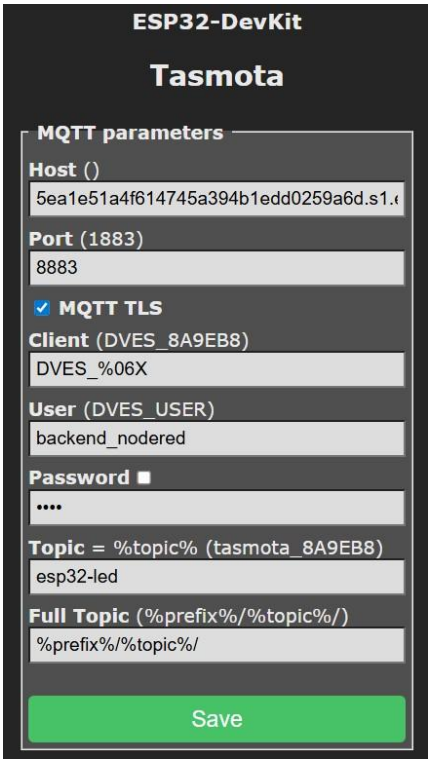
Licht Übergang/Fade
Topic: cmnd/esp32-led/Fade
Payload/Message: on **ODER** off
=> Lichter müssen anschließend erneut gesetzt werden, um Effekt zu sehen (Ändert nur Übergang zwischen Farben)

Licht Geschwindigkeit/Speed

Topic: cmnd/esp32-led/SPEED
Message/Payload: Anzahl in Sekunden
=> Beispielsweise Message/Payload: 3
(In 3 Sekunden werden die Lichter angezeigt)
=> Funktioniert nur mit Zusatzeffekten z.B Fade!

Alle Lichter steuern
Topic: cmnd/esp32-led/Color
Payload/Message: <RGB/Hex>
=> Besipiel Payload/Message: 0,255,0

MQTT Konfiguration des ESP-32:



MQTT-Broker Kram

HiveMQ Broker URL:

5ea1e51a4f614745a394b1edd0259a6d.s1.eu.hivemq.cloud

Port: 8883 Websocket-Port: 8884

PERMISSON FÜR NUR SUB:

User: testuser

Passwort: O#%~54{Kf{-c-7t

PERMISSION FÜR SUB+PUB:

User: backend_nodered

Passwort: vzd)4S=5FZJ(U!5

Für später und Doku, da wo wir offiziell hinwollen:

Die Kommunikation erfolgt verschlüsselt über MQTT TLS (Port 8883). Der Zugriff auf den Broker ist durch individuelle Benutzerkonten und rollenbasierte Topic-Berechtigungen abgesichert.

Manuelles ESP-32 Flashen mit Tasmota über Windows Terminal

Vorbereitung

1. Downloade die passende Firmware für die ESP-32: (In unserem Fall die tasmota32.factory.bin)

Link: https://ota.tasmota.com/tasmota32/

2. Sollte der USB Port, an der die ESP32 hängt, nicht erkannt werden, muss zusätlich der USB-Bridge Driver heruntergeladen werden

Link: https://www.silabs.com/software-and-tools/usb-to-uart-bridge-vcp-drivers?tab=downloads
Downloade dann den ***CP210x Universal Windows Driver*** Treiber.

3. Installieren von Python

Link: https://www.python.org/ftp/python/3.14.4/python-3.14.4-amd64.exe

Durchführung

Um nun mit dem manuellen Flashen von Tasmota zu beginnen, muss zunächst das Python Paket "esptool" heruntergalden und installiert werden. Hierfür gib folgenden Befehl im Windows Terminal ein:

```
py -m pip install esptool
```

Nun müssen wir in das Verzeichnis wechseln, in der die Tasmota Firmware liegt (endend mit factory.bin)

```
cd %USERPROFILE%\Downloads
```

Bevor wir die ESP-32 flashen, sollten wir die aktuellen Daten der ESP-32 löschen, um Datenrendundanz zu vermeiden:

```
py -m esptool --chip esp32 --port COM3 erase_flash
```

COM3 = USB Port --> Kommt drauf an, an welchem Port die ESP-32 angeschlossen ist
=> Das kann man unter dem "**Geräte Manager**" in Windows herausfinden

Nun flashen wir die ESP-32 mit Tasmota:

```
py -m esptool --chip esp32 --port COM3 write_flash 0x0 tasmota32.factory.bin
```

Tasmota Webkonsole aufrufen:

Die Tasmota Webkonsole, kann über folgendem Link erreicht werden:

Link: https://tasmota.github.io/install/
Sollte die ESP-32 noch nicht mit dem WLAN verbunden sein, so klicke auf "**Change Wi-Fi**" und anschließend auf "**Visit Device**".

Sollte die ESP-32 sich noch in einem anderen Netz befinden, so kann man auch über die Webkonsole die ESP-32 mit einem anderen WLAN-Netz verbinden.
Dazu geht man unter "**Logging & Console**", nachdem man den Tasmota Webinstaller offen hat und gibt in die Konsole folgendes ein:

Backlog SSID1 <WLAN SSID>; Password1 <PASSWORT>

RFID Konfiguration

Verkabelung:

Pinanbindung des RFID Sensors zur ESP-32:
(Wir schauen auf die ESP-32 von oben mit dem USB-C Port nach unten gerichtet)

SDA --> IO05 (linkes Pinbrett 7. Pin innen)
SCK --> IO018 (linkes Pinbrett 4. Pin innen)
MOSI --> IO023 (linkes Pinbrett 6. Pin innen)
MISO --> IO019 (linkes Pinbrett 5. Pin innen)
GND --> GND (rechtes Pinbrett 1. Pin außen)
RST --> IO022 (rechtes Pinbrett 3. Pin innen)
3.3V --> 3.3V (linkes Pinbrett 8. Pin innen)

Tasmota GPIO-Konfiguration:

Anmeldung zur Webkonsole der RFID-ESP32:
Nutzername: *admin*
Passwort: *wmzbj3JNAi1D0vwU1QGX*

Nun konfigurieren wir die einzelnen Pins auf der ESP 32
=> Configuration > Module

GPIO018 --> **SPI CLK**
GPIO019 --> **SPI MISO**
GPIO023 --> **SPI MOSI**
GPIO05 --> **RC522 CS**
GPIO022 --> **RC522 Rst**

Broker-Topic:

Die ganzen RFID Scans werden unter dem Topic **esp32-rfid** veröffentlicht!
=> Der Output kommt dann unter dem Topic **tele/esp32-rfid/SENSOR** an!

Credentials

Tasmota Webkonsole: ESP32 für LED:
Nutzername: admin
Webpasswort: h1NVITgqxwccLfY0Wx2I

Tasmota Webkonsole: ESP32 für RFID:
Nutzername: admin
Webpasswort: wmzbj3JNAi1D0vwU1QGX

Node-Red:
Domäne: nodered.burgerbrater.de
Nutzername: admin
Passwort: w1W54GRFY1M014c

HiveMQ Creds:
Name: burgerbrater@proton.me
Passwort:)@3mvNF)8YU-US/7&=,'

HiveMQ backend User:
Nutzername: backend_nodered
Passwort: vzd)4S=5FZJ(U!5

PostgresSQL:
Datenbank: access_control
Tabelle: rfid_users
User: nodered
Passwort: Dg3fVpjvfhzb0ie

Host intern: postgres
Port intern: 5432

HiveMQ Publish ONLY
Username: broker_pub
Passwort: cx7R0fUu87Ljf61N0NLR

HiveMQ Subscribe ONLY
Username: broker_sub
Passwort: LrCqrtu9qpe8uJIyKeOB

Doku und Präsi

Zur Doku:

FERTIG !!!

Zur Präsi:

Mein Vorschlag zum Präsentieren:

Themenblock	Referenten	Inhalt	Zeit
Vorstellung	Daniel	Begrüßung, Teamvorstellung, Start der Präsentation	ca. 1 min.
Übersicht Gesamtprojekt	offen? alle/einer?	Was ist unser Projekt (ganz kurz: Beschreibung / Ziel)? Grafik Gesamtarchitektur zeigen und erklären	ca. 1-2 min. ca. 2 min.
LIVE DEMO	alle ihr/einer?	Unterbrechung der Präsi für Live-Demo... Beschreibung welche Komponenten und grob Ablauf	ca. 3-5 min.
Grundlagen / Einführung	Daniel	ggf. nochmal allgemeine Worte? Kurze Theorie zu RFID	ca. 2-4 min. ?
ESPs, MQTT	Lars	individueller Part	max. 5 min. ?
Backend, Dienste	Daniel	individueller Part	max. 5 min. ?
Node-RED, Flows	Phil	individueller Part	max. 5 min. ?
YOLO-Cam	Nico	individueller Part	max. 5 min. ?
Fazit	Daniel	Ergebnis, Verlauf (Probleme, Lösungen?), Ausblick Beendigung der Präsi :)	ca. 2 min.

Idee / Option: **Jeder beginnt** seine individuellen Folien kurz **mit dem Schaubild** und ordnet ein, wo man sich gerade befindet und worum es technisch geht ?!?

Offen: **Übergänge moderierend** gestalten (nicht: "Jetzt kommt Lars." :D, sondern Überleitungssatz zu Thema, ohne Namen zu nennen) **ODER direkt** mit Blickkontakt absprechend und nahtlos übernehmen ?!? :D ...

Offen bei Live-Demo: Klare Rollenverteilung vorab. Wer hält Chip? Wer redet/erklärt? Wer läuft durch Cam?

Mein Vorschlag zur Bearbeitung:

An Tabelle orientieren für Ablauf. **Jeder bearbeitet seinen individuellen Part** auf Basis der Vorlagen-Präsi-Datei vor und lädt sie hier hoch.

Ausstehend: Wer trägt unterschiedliche Dateien **in finale Datei zusammen?** Wer hat sie als . **odt und .pdf?** Wer hat **Lapi** / präsentiert, etc.?

Bilder