Dokumentation: Modulares BLE-Radar Türöffnungssystem

Stand: 10. November 2025
Version: 4.0

1. Einleitung und Projektübersicht

Dieses Dokument fasst den aktuellen Diskussionsstand, die Architekturentscheidungen und die technische Implementierung für ein automatisiertes Türöffnungssystem zusammen.

Das System basiert auf der Fusion von **mmWave-Radar-Technologie** zur präzisen Bewegungserkennung und Trendanalyse sowie **BLE-Beacons** zur Personenidentifikation.

Ziel: Schaffung eines robusten, schnellen und diskreten Zutrittskontrollsystems, das die Tür nur für berechtigte Personen öffnet, die mit einer klaren Eintrittsabsicht auf das Gebäude zukommen. Fehlertoleranz (ein "vergebliches Summen" ist akzeptabel) und Fehlervermeidung (die Tür *muss* für Berechtigte aufgehen) haben Priorität vor absoluter Sicherheit. Diskretion ist wichtig: Das System soll Besuchern nicht anzeigen, dass ihnen der Zugang verweigert wurde.

Die Entwicklung erfolgt modular in Python auf einem Raspberry Pi, basierend auf asyncio für eine effiziente, nebenläufige Verarbeitung von Sensor-I/O und Logik.

2. Herleitung der modularen Architektur

Ausgangspunkt war ein monolithisches Python-Skript (BLE_tueroeffner.py), das BLE-Scanning, Display-Steuerung und Türöffnungslogik in einem einzigen Thread vereinte. Dies führte mit wachsender Komplexität zu Wartbarkeitsproblemen und blockierenden Operationen, die die Reaktionszeit des Systems beeinträchtigten.

2.1. Motivation für die Radar-Integration

Die ursprüngliche, rein BLE-basierte Distanzschätzung (über RSSI - Received Signal Strength Indicator) erwies sich als zu unzuverlässig für eine präzise Triggerung der Türöffnung. Schwankungen im Signal führten zu Fehlauslösungen oder verzögertem Öffnen.

Die Einführung eines mmWave-Radarsensors (zunächst Seeedstudio RD-03D, später auch HLK-LD2450) behebt dieses Problem fundamental. Der Radarsensor liefert kontinuierlich präzise Daten über:

- **Distanz (Y)** zum Sensor in Millimetern.
- **Seitliche Position (X)** in Millimetern.

- **Geschwindigkeit** in cm/s (inklusive Bewegungsrichtung relativ zum Sensor).

Dies führte zu der fundamentalen Architekturentscheidung, die **Radar-Logik zum "Master"** der Türöffnungsentscheidung zu machen. BLE dient nur noch der Verifikation ("Wer ist das?"), während das Radar bestimmt "Was tut er?" und "Wann genau soll geöffnet werden?".

2.2. Das aktuelle Modulmodell (Stand November 2025)

Um das System wartbar und erweiterbar zu halten, wurde es in spezialisierte, asynchron arbeitende Module aufgeteilt:

- **M_TuerOeffner_R.py (Orchestrator):** Das Hauptprogramm. Es ist schlank gehalten und verantwortlich für:
  - Initialisierung aller Subsysteme (Radar, Display, BLE-Datenstrukturen).
  - Starten der asynchronen Haupt-Tasks (radar_reader_task, radar_logic_task, display_manager_task).
  - Überwachung des Lebenszyklus der Tasks.
  - Sauberes Herunterfahren (Graceful Shutdown) aller Komponenten bei Fehlern (z.B. Radar-Verlust) oder beabsichtigter Beendigung (SIGINT).
- **radar_logic.py (Entscheidungszentrale):** Das Herzstück des Systems. Es implementiert die komplexe Entscheidungslogik als asynchrone State Machine. Architektonisch ist es weiter unterteilt, um I/O von Logik zu entkoppeln:
  - **Reader Task:** Ein dedizierter Endlos-Task, der kontinuierlich Daten von der Radar-Hardware liest und den neuesten vollständigen Frame in eine interne Queue schiebt.
  - **Logic Task:** Verarbeitet die Daten aus der Queue, führt die Trendanalyse durch, verwaltet die Zustände (IDLE, TRACKING, COOLDOWN), triggert bei Bedarf den BLE-Scan und trifft die finale Entscheidung zum Öffnen der Tür.
- **ble_logic_R.py (Identifikations-Service):** Bietet eine "On-Demand"-Funktion für BLE-Scans. Es scannt nicht kontinuierlich, sondern nur auf explizite Anforderung durch die Radar-Logik für eine begrenzte Zeit (z.B. 1.5s). Es prüft gefundene Beacons gegen eine interne Datenbank (beacon_identification_state) autorisierter Nutzer.
- **display_logic.py (HMI - Human Machine Interface):** Verwaltet das ePaper/Sharp-Display.
  - Zeigt standardmäßig Uhrzeit, Datum und aktuelle Wetterdaten (Temperatur, Wind, Niederschlag) an, die es periodisch von einer PWS-API (Personal Weather Station) abruft.
  - Empfängt über eine globale Queue (display_status_queue) Statusmeldungen von der Radar-Logik (z.B. "ACCESS_GRANTED") und überlagert diese temporär als Icons (z.B. Schlüssel-Symbol).
- **door_control.py (Aktor-Service):** Kapselt die physische Ansteuerung des Türöffners. Aktuell wird dies über den Aufruf des externen Tools codesend realisiert, das ein 433MHz-Signal an ein Funkrelais sendet. Es verwaltet intern einen Cooldown-Timer, um eine Überlastung des Relais durch zu häufige Schaltvorgänge zu verhindern.

- **config.py (Konfigurations-Manager):** Lädt die zentrale Konfigurationsdatei system_config.json beim Start. Stellt die Parameter allen Modulen über eine get()-Hilfsfunktion zur Verfügung. Definiert auch zentrale Hardware-Parameter, die nicht in der JSON stehen (wie GPIO-Pin-Belegungen für das Display).
- **globals_state.py (Geteilter Laufzeit-Status):** Enthält Variablen, die zwingend zwischen Modulen geteilt werden müssen. Aktuell sind dies primär die display_status_queue für die Kommunikation zwischen Radar- und Display-Logik sowie eine globale cleanup_gpio-Funktion für das saubere Beenden.

3. Physik der Umgebung und Koordinatensystem (Level 0)

Das Verständnis der physischen Installation ist kritisch für die Logik der Bewegungserkennung.

3.1. Physischer Aufbau

- **Ort:** Der Sensor befindet sich in einer gedruckten Box an der Außenkante eines Briefkastens.
- **Position:** Halbe Höhe der Treppe, unter einem Vordach.
- **Blickrichtung:** Der Sensor "schaut" nach Osten auf die Treppe. Die Treppe selbst führt rechtwinklig vom Zugangsweg (der von Norden kommt) nach Osten zur Haustür.
- **Winkel:** Der Sensor ist so montiert, dass er ca. 34° aus dem nördlichen Blickwinkel der Wand heraus "schaut". Er deckt damit den Zugangsweg schräg ab.
- **Material:** Die Box besteht aus PETG. Der Radarsensor hat ca. 2mm Plastik "vor sich", was für mmWave-Wellen transparent ist.

3.2. Koordinatensystem und Bewegungsablauf

Aus dieser Montage ergibt sich folgendes Koordinatensystem für den Sensor:

- **Y-Achse (Distanz):** Zeigt vom Sensor weg. Ein Objekt auf dem Zugangsweg nähert sich dem Sensor, d.h. der Y-Wert verringert sich kontinuierlich.
- **X-Achse (Seitlich):** Zeigt (je nach genauer Montage, konfigurierbar als expected_x_sign) quer zum Zugangsweg.

Typischer Bewegungsablauf bei Eintritt:

1. **Annäherung:** Person kommt von Norden den Weg entlang.
    - Radar: Y-Wert nimmt stetig ab. X-Wert ist konstant positiv (oder negativ). Geschwindigkeit ist negativ (auf Sensor zu).
2. **Abbiegen:** Kurz vor dem Sensor biegt die Person nach rechts (Osten) zur Treppe ab.
    - Radar: Y-Wert erreicht ein Minimum (ca. 200-500mm).
3. **Eintritt:** Die Person steigt die Treppe hoch und passiert dabei den Sensor.

- Radar: **Vorzeichenwechsel der X-Achse!** Die Person kreuzt die virtuelle Mittellinie des Sensors (Y-Achse) von "vor dem Sensor" nach "hinter den Sensor" (aus Sicht des Treppenaufgangs).
- **Dies ist der geometrisch optimale Trigger-Zeitpunkt für die Türöffnung.**

4. Logischer Ablauf und Implementierung (Radar Logic - Level 1 & 3)

Die Kernlogik in radar_logic.py ist als **State Machine** implementiert, um diesen physischen Ablauf robust abzubilden.

4.1. Zustände der State Machine

1. **IDLE:** Das System wartet passiv auf ein Objekt.
2. **TRACKING:** Ein Objekt wurde erkannt. Die Radardaten werden aktiv in eine Historie (Rolling Buffer der letzten N Frames) geschrieben und analysiert.
3. **COOLDOWN:** Nach einer Türöffnung wird das System für eine konfigurierbare Zeit (z.B. 3s) "taub" geschaltet, um Mehrfachauslösungen durch dieselbe Person (die noch im Erfassungsbereich steht) zu verhindern.

4.2. Zweischnittige Analyse (im TRACKING-Zustand)

Um eine hohe Robustheit gegen Fehlauslösungen (z.B. vorbeilaufende Personen, Haustiere, wackelnde Büsche) zu erreichen, müssen zwei Analysen sequenziell "grünes Licht" geben:

Block A: Trendanalyse & Identifikation ("Wer ist das und was will er?")

- **Ziel:** Frühzeitige Erkennung einer *ernsthaften* Annäherung (kein Rauschen) und Verifikation der Berechtigung.
- **Implementierung (Trend):** Statt nur Momentanwerte zu prüfen, wird eine lineare Regression (Least Squares) über die Y-Positionen der letzten history_size Frames (z.B. 7) berechnet.
  - Ist die Steigung negativ und signifikant stärker als das konfigurierte Rauschen (speed_noise_threshold), wird der interne Status auf IntentStatus.KOMMEN gesetzt.
  - Ist die Steigung positiv (Person entfernt sich), wird IntentStatus.GEHEN gesetzt und die State Machine kehrt zu IDLE zurück.
- **Implementierung (Identifikation):** Sobald der Trend "KOMMEN" signalisiert, wird *einmalig* pro Tracking-Zyklus ein asynchroner BLE-Scan gestartet.
  - Wird ein autorisierter Beacon gefunden, wird der interne Status auf BLEStatus.SUCCESS gesetzt.
  - Läuft der Scan ohne Treffer ab, wird BLEStatus.FAILED gesetzt und die State Machine kehrt zu IDLE zurück.

Block B: Trigger-Zeitpunkt ("Wann genau öffnen?")

- **Voraussetzung:** Block A muss vollständig erfolgreich sein (Trend = KOMMEN **UND** BLE = SUCCESS).
- **Ziel:** Den ergonomisch besten Moment für die Öffnung finden (genau dann, wenn die Person zur Tür abbiegt).
- **Kriterium ("Akuter Vorzeichenwechsel"):** Das System vergleicht die X-Position des aktuellen Frames mit dem vorletzten Frame.
    - Findet ein Vorzeichenwechsel statt (z.B. von +X auf -X), bedeutet dies, die Person hat die Mittellinie überschritten.
    - Um Fehltrigger durch Rauschen bei weit entfernten Objekten zu vermeiden, ist dieser Trigger nur gültig, wenn die Y-Distanz kleiner als sign_change_y_max (z.B. 500mm) ist.
- **Aktion:** Wenn der Trigger gültig ist -> Sende Öffnungsbefehl -> Wechsle in COOLDOWN.

5. Variablen-Dokumentation

5.1. Konfigurations-Variablen (system_config.json)

Diese Variablen steuern das statische Verhalten des Systems und werden beim Start geladen.

Kategorie	Variable	Beschreibung & Typischer Wert	Verwendet in Modul
system_globals	ibeacon_uuid	UUID der zu suchenden iBeacons. (z.B. E2C56DB5...)	ble_logic_R
	eddystone_namespace_id	Namespace ID für Eddystone-UID Beacons.	ble_logic_R
	relay_activation_duration_sec	Dauer des Öffnungsimpulses in Sekunden (z.B. 4).	door_control
	min_detection_interval	Globaler Cooldown für den Aktor in Sekunden (z.B. 20). Verhindert Relais-Überlastung.	door_control
weather_config	station_id	ID der Personal Weather Station (PWS) für lokale Daten.	display_logic
	api_key	API-Key für Weather Underground.	display_logic
	query_interval_sec	Zeit zwischen zwei Wetter-API-Abfragen in	display_logic

Kategorie	Variable	Beschreibung & Typischer Wert	Verwendet in Modul
		Sekunden (z.B. 300 = 5min).	
logging_config	level	Log-Level (z.B. INFO, DEBUG, TRACE).	config, alle
	file_enabled	Ob Logs in eine Datei geschrieben werden (true/false).	config
	file_path	Pfad zur Logdatei (z.B. tuer_oeffner.log).	config
radar_config	uart_port	Serielle Schnittstelle des Sensors (z.B. /dev/ttyAMA2).	radar_logic
	ble_scan_max_duration	Max. Dauer eines On-Demand BLE-Scans in Sekunden (z.B. 1.5).	radar_logic
	speed_noise_threshold	Min. Geschwindigkeit (cm/s), um ein Target als "bewegt" zu akzeptieren (z.B. 5).	radar_logic
	expected_x_sign	Erwartete X-Richtung bei Annäherung (positive oder negative).	radar_logic

Kategorie	Variable	Beschreibung & Typischer Wert	Verwendet in Modul
		Montageabhängig.	
	door_open_comfort_delay	Optionale Verzögerung zwischen Trigger und tatsächlicher Öffnung in Sekunden (z.B. 0.5).	radar_logic
	cooldown_duration	System-Pause nach Ereignis in Sekunden (z.B. 3.0).	radar_logic
	history_size	Größe des gleitenden Fensters für die Trendanalyse (z.B. 7 Frames).	radar_logic

	sign_change_y_max	Max. Y-Distanz (mm) für einen gültigen Trigger. Filtert ferne Vorbeiläufer (z.B. 500).	radar_logic
	sign_change_x_max	Max. seitlicher Abstand (mm) bei X=0 Durchgang. Filtert Rauschen (z.B. 700).	radar_logic
	radar_loop_delay	Pause zwischen I/O-Zyklen in Sekunden (z.B. 0.05 = 50ms). Steuert die Abtastrate.	radar_logic
known_beacons	*(Array)*	Liste autorisierter Beacons mit MAC, Name, IDs und Berechtigungsstatus.	ble_logic_R
auth_criteria	*(Objekt)*	Definiert, welche Beacon-Merkmale (iBeacon, UID, URL, MAC) für eine erfolgreiche Identifikation REQUIRED sind.	ble_logic_R

5.2. Interne Konstanten (Hardcoded)

Diese Werte sind fest im Code verankert, da sie sich selten ändern oder hardware-spezifisch sind.

Modul	Konstante	Wert / Bedeutung
config.py	DISPLAY_WIDTH / HEIGHT	400 / 240 (Auflösung des Sharp Displays).
config.py	SHARP_*_PIN	GPIO-Pin-Definitionen für das Display (D6, D5, D22).
config.py	CODESEND_PATH	/usr/local/bin/codesend (Pfad zum 433MHz-Sendetool).
config.py	CODESEND_MIN_DURATION	3 (Minimale Impulsdauer in Sekunden, die codesend akzeptiert).
rd03d_async.py	BAUDRATE	256000 (Feste Baudrate des RD-03D Sensors).
ld2450_async.py	BAUDRATE	256000 (Feste Baudrate des LD2450 Sensors).

5.3. Wichtige Laufzeit-Zustandsvariablen

Diese Variablen ändern sich dynamisch während des Betriebs und bilden den aktuellen Systemzustand ab.

Variable (Objekt)	Bedeutung / Mögliche Werte	Speicherort
_state.system_state	Hauptzustand der State Machine: IDLE, TRACKING, COOLDOWN.	radar_logic.py
_state.intent_status	Ergebnis der Trendanalyse	radar_logic.py

	(Block A): NEUTRAL, KOMMEN (Annäherung), GEHEN (Entfernung).	
_state.ble_status	Status der Identifikation: UNKNOWN, SCANNING, SUCCESS (berechtigt), FAILED.	radar_logic.py
_state.history	deque-Ringpuffer der letzten N Radar-Messwerte (Zeitstempel, x, y) für die Trendanalyse.	radar_logic.py
beacon_identification_state	Dictionary (Cache) für alle bekannten Beacons, deren Konfiguration und letzten Sichtkontakt.	ble_logic_R.py (via globals_state noch)
last_successful_weather_data	Dictionary (Cache) für die zuletzt abgerufenen Wetterdaten, um API-Calls zu minimieren.	display_logic.py (via globals_state noch)
_last_codesend_time	Zeitstempel (Float) der letzten Aktor-Auslösung zur Einhaltung des globalen Cooldowns.	door_control.py (via globals_state noch)
display_status_queue	asyncio.Queue für Statusmeldungen (z.B. {"type": "status", "value": "ACCESS_GRANTED"}) von Radar an Display.	globals_state.py

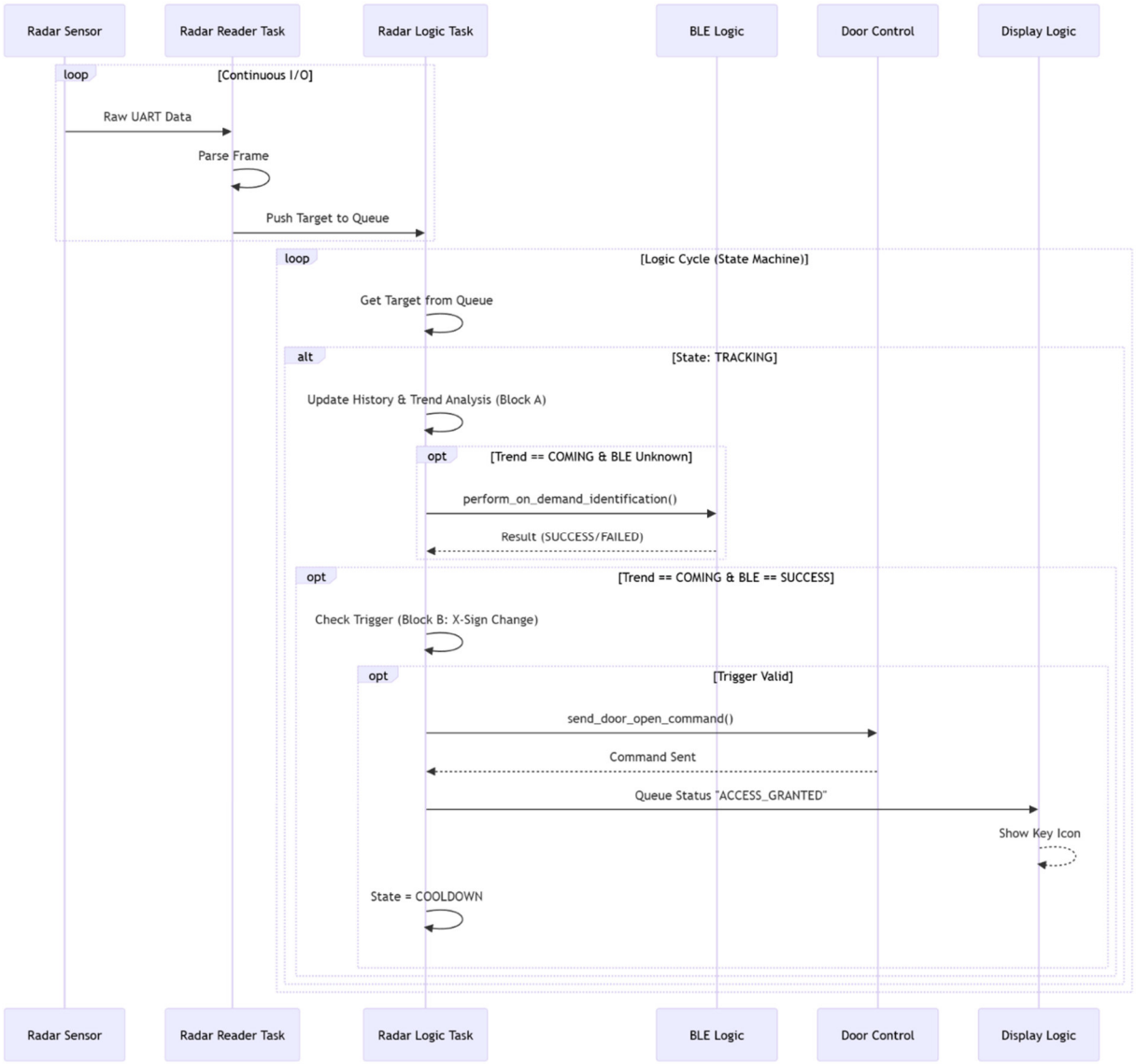
6. Hardware-Abstraktion

Das System ist so konzipiert, dass der Radarsensor austauschbar ist.

- **Treiber-Schicht:** Die Module rd03d_async.py und ld2450_async.py implementieren die spezifischen UART-Protokolle der jeweiligen Sensoren.
- **Einheitliche Schnittstelle:** Beide Treiber stellen eine Klasse Target bereit, die die rohen Sensordaten in ein einheitliches Format normalisiert:
  - x: Seitliche Position in mm.
  - y: Distanz in mm.
  - speed: Geschwindigkeit in cm/s (negativ = Annäherung).
- **Konfiguration:** Die Auswahl des aktiven Treibers erfolgt aktuell noch hardcoded im Header von radar_logic.py über die Variable SENSOR_TYPE ("RD03D" oder "LD2450").

7. Flow-Diagramme

7.1. System-Interaktion



7.2. Radar State Machine & Entscheidungslogik