ionpy Framework: Erweiterte Architektur-Spezifikation (Vollständig)

TODO

- Checkn ob Webseite Reconnect macht wenn man ionpy neu startet
- Inventar über Web View einstellbar machen
  - Inventar ist nicht bestandteil vom Projekt (Idee vielleicht die Inventar.yaml mit im Projekt speichern → schnelles wiederherstellen ??)
  - Gerät hinzufügen, löschen, Ändern, …
- Settings nicht überschreiben beim Einstellen → siehe Punkt 1
- Projekt anlegen / speichern / laden → Frontend
  - Speichern wo ? Frontend oder Backend ????
- RP2350 als Testgerät
- Buttons mit X Settings ausführen

Dieses Dokument beschreibt die integrale Architektur-Erweiterung des ionpy-Frameworks. Es dient als verbindliche Grundlage für die Implementierung neuer Entitätstypen, haptischer Steuerungen und geräteübergreifender Automatisierung.

1. Dynamische Eingabesynchronisation (Race Condition Schutz)

Um zu verhindern, dass Hintergrund-Polling Benutzereingaben im Frontend überschreibt, wird ein duales Sperrsystem implementiert.

1.1 Backend: Mute-Timer (AbstractDevice)

[cite_start]In der Klasse `AbstractDevice` (hardware/base.py) wird eine zeitbasierte Sperre pro Entität eingeführt[cite: 117, 120].

- [cite_start]**Mechanismus**: Ein Dictionary `self._last_command_time: Dict[str, float]` speichert den Zeitpunkt des letzten Schreibbefehls[cite: 11, 54].
- **Logik**:
  - [cite_start]Sobald `execute_command()` aufgerufen wird, erhält die `entity_id` einen Zeitstempel[cite: 13, 138].
  - [cite_start]Die Methode `update_entity()` prüft diesen Zeitstempel: Liegt er weniger als **3,0 Sekunden** in der Vergangenheit, wird das Sample verworfen und nicht auf den Bus publiziert[cite: 87, 136].
- **Ziel**: Die Hardware hat Zeit, den Wert intern zu setzen, und der Polling-Loop liest keine “alten” Werte mehr zurück, während der User noch interagiert.

1.2 Frontend: Universeller Focus-Lock (JS)

In der Web-UI (settings.html) wird eine automatische Erkennung aktiver Eingabefelder implementiert.

- **Mechanismus**: Nutzung eines `Set()` namens `activeInputs`.
- **Event-Delegation**:
  - `focusin`: Fügt die Element-ID zum Set hinzu.
  - `focusout`: Entfernt die ID nach einer kurzen Verzögerung (ca. 300-500ms).
- **WebSocket-Logik**: Die Funktion `channel.onmessage` prüft vor dem Update eines HTML-Elements, ob dessen ID im Set vorhanden ist. [cite_start]Falls ja, wird das Update verworfen[cite: 53, 54].

Neue View Multisensor Device

- auf einen Schlag alle Readings anzeigen
- Checkbox on/off für MAIN = “main”, CONFIG = “config”, DIAGNOSTIC = “diag”
- Device basiert

2. Strukturierte Daten: TableEntity (Deep Dive)

Die `TableEntity` ist das Herzstück für komplexe Geräteeigenschaften wie Speicherplätze (M1-M10), Profil-Listen (Sequenzer) oder Zell-Übersichten.

2.1 Datenstruktur & Schema

Eine `TableEntity` kapselt nicht nur Daten, sondern auch deren Bedeutung.

- **Schema (columns)**: Definition der Spalten-Metadaten.
  - Jede Spalte definiert: `key`, `name`, `type` (number/text/toggle/action), `unit`, sowie Constraints (`min`, `max`, `step`).
- **Daten (value)**: Eine Liste von Dictionaries, wobei jedes Dictionary eine Zeile darstellt.
- **Typen**: Unterscheidung zwischen `fixed_size` (z.B. genau 10 Speicherplätze) und `dynamic_size` (Zeilen hinzufügbar/löschbar).

2.2 Erweiterte Interaktions-Logik

- **Row-Updates**: Das Frontend sendet Koordinaten-Pakete: `{ “row”: r, “col”: “key”, “val”: value }`.
- **Atomic Row Actions**: Unterstützung einer Spalte vom Typ `button`. Dies ermöglicht “Apply”-Buttons pro Zeile, um einen kompletten Parametersatz (z.B. Volt und Ampere eines Presets) gleichzeitig an die Hardware zu senden, um instabile Zustände zu vermeiden.
- **Active Row Tracking**: Ein zusätzliches Attribut `active_row_index` markiert die Zeile, die das Gerät aktuell tatsächlich verwendet (z.B. welcher Speicherplatz gerade geladen ist).
- **Zell-basiertes Muting**: Die Mute-Logik aus Kapitel 1 wird auf Zellebene angewendet, sodass eine Bearbeitung in Zeile 1 nicht die Live-Updates von Zeile 2 blockiert.

Software Liste

- Abspulen von Lastprofilen für PSU / Senke → rein Softwar5e basiert

Settings auf Button legen

- **neue View** !
- Buttons Beschriftung setzen (Größe ??)
- (mehrere) Aktions hinterlegen für Device
- ggf. inkl. Start ??
- Beispiel → PSU → 12V Setzen, max 1A, OCP an, Output ON

3. Gamepad-Integration (HID-Steuerung)

Haptische Steuerung via USB-Controller, realisiert durch das `pygame`-Subsystem.

3.1 GamepadManager (hardware/system/gamepad.py)

Ein neuer Treiber-Typ, der autonom nach Controllern sucht.

- [cite_start]**Discovery**: Nutzt `pygame.joystick.get_count()` und `get_id()`, um Controller dynamisch zu finden, ohne Hardcoding in der Config[cite: 63, 64].
- [cite_start]**GamepadEntity**: Eine neue Entitätsklasse, die den Zustand (Axes, Buttons, Hats, Triggers) als Snapshot-Objekt im `value`-Feld hält[cite: 135].

3.2 Haptisches Feedback & Visualisierung

- [cite_start]**UI-Widgets**: Spezielle Web-Komponenten für Joysticks (Fadenkreuz) und Trigger (Druckempfindliche Balken via `UIMode.LEVEL`)[cite: 422].
- **Sicherheitskonzept**: Implementierung eines “Deadman-Switch” (Totmannknopf). Steuerbefehle werden nur an andere Geräte weitergeleitet, wenn eine definierte Taste am Gamepad gehalten wird.

4. LogicService: Die Automation Bridge

[cite_start]Zentraler asynchroner Dienst in der `SystemEngine`[cite: 81], der als Vermittler zwischen dem Bus und den Geräte-Kommandos fungiert.

4.1 Die Rule-Engine

[cite_start]Der Dienst abonniert den `EventBus` [cite: 85, 427] und prozessiert Regeln aus einer `rules.json`.

- [cite_start]**Trigger**: Ein Sample von Gerät A (z.B. Gamepad-Achse oder BMS-Temperatur)[cite: 424].
- **Transformation (Scaling)**: Mathematische Umwandlung von Eingangswerten (z.B. Gamepad-Stick -1.0…+1.0) in Zielwerte (z.B. Netzteil 0.0…32.0 V).
- [cite_start]**Action**: Ausführung von `engine.execute_command(target_dev, key, transformed_val)`[cite: 84, 433].

4.2 Cross-Device Szenarien (Beispiele)

- **Synchronisation**: Die elektronische Last (Senke) folgt automatisch der Spannung des Netzteils (Quelle), um eine konstante Leistung (CP-Mode) über das Framework zu simulieren.
- **Master-Slave**: Zwei Netzteile werden so gekoppelt, dass Kanal 2 immer exakt dem Wert von Kanal 1 folgt.

5. Erweiterter Entitäten-Katalog

[cite_start]Zusätzliche spezialisierte Typen für professionelle Laboranforderungen[cite: 421, 422, 423]:

Typ	UI-Repräsentation	Funktionalität
LogEntity	Scrollende Konsole	Lokaler Ereignis-Speicher für gerätespezifische Fehler (z.B. SCPI-Fehlermeldungen).
StatusIndicator	Virtuelle LED	Farb-Mapping für Zustände (z.B. 0=Off, 1=OK/Grün, 2=Warnung/Gelb, 3=Alarm/Rot-Blinkend).
XYGraphEntity	Kennlinien-Plot	Darstellung von X-Y-Beziehungen (z.B. Batterie-Entladekurve: Spannung über Kapazität).
FileEntity	Upload/Download	Schnittstelle für Firmware-Dateien (z.B. ESPHome .bin) oder Konfigurations-Exports.
RangeEntity	Multi-Slider/Input	Gruppiert logisch zusammengehörige Werte für Sweeps (Start, Stop, Step, Intervall).
ScheduleEntity	Zeitplan-Editor	Verwaltung von Zeitereignissen (z.B. “Schalte Ausgang an Wochentagen um 08:00 Uhr an”).

6. Implementierungs-Leitfaden für KI-Entwicklung

- [cite_start]**Concurrency**: Alle Logik-Operationen müssen asynchron (`async/await`) ausgeführt werden, um den Hardware-Poll-Loop nicht zu blockieren[cite: 1, 9].
- [cite_start]**Caching**: Die `SystemEngine` nutzt ihren `state_cache` als “Single Source of Truth” für die Logic-Regeln[cite: 81, 84].
- **Modularität**: Neue Entitäten müssen in `structures/entities.py` definiert und in der `settings.html` mit einem entsprechenden UI-Generator verknüpft werden.

7. Erweiterte Web-Views (Advanced Visualization)

Um die wachsende Komplexität der Daten (Gamepad, BMS, IMU-Sensoren) beherrschbar zu machen, werden spezialisierte Views implementiert.

7.1 XYZ / 3D-Visualisierung (Spatial View)

Diese View nutzt Bibliotheken wie **Three.js** oder **Plotly.js**, um Daten im dreidimensionalen Raum darzustellen.

- [cite_start]**Anwendungsfall A: IMU/Orientierung**: Visualisierung der Fluglage oder Position eines Sensors (basierend auf der `VectorEntity` [cite: 419]). [cite_start]Ein 3D-Modell (z.B. ein PCB oder eine Batteriebox) neigt sich in Echtzeit entsprechend der Daten aus dem `VirtualLaboratory`[cite: 252, 263].
- **Anwendungsfall B: Multi-Parameter-Sweeps**: Darstellung von Abhängigkeiten wie "Effizienz über Eingangsspannung und Laststrom". Hierbei entsteht eine 3D-Oberfläche (Surface Plot).
- **Anwendungsfall C: Raum-Mapping**: Wenn Sensordaten mit Positionsdaten gekoppelt sind (z.B. Temperatur-Mapping einer Oberfläche).

7.2 Multi-Device Dashboard (Global View)

[cite_start]Die aktuelle UI ist stark auf einzelne Tabs pro Gerät fokussiert[cite: 16]. Die `Global View` bricht diese Struktur auf.

- **Konzept**: Eine frei konfigurierbare Kachel-Ansicht (Grid-Layout), in der Entitäten verschiedener Geräte gemischt werden können.
- [cite_start]**Beispiel**: Ein "Power-Dashboard", das die Eingangsleistung vom `UDP3305` [cite: 303][cite_start], den Zellstatus vom `JbdBms` [cite: 145] [cite_start]und die Lastdaten der `AtorchDL24` [cite: 211] auf einer einzigen Seite zusammenfasst.

7.3 Logic-Flow Visualizer (Automation View)

Da der geplante `LogicService` komplex werden kann, ist eine textuelle Regel-Liste oft unübersichtlich.

- **Konzept**: Eine Node-basierte Darstellung (ähnlich wie Node-RED).
- [cite_start]**Darstellung**: Blöcke repräsentieren Trigger (z.B. Gamepad [cite: 255][cite_start]), Logik-Gatter und Aktionen (z.B. Netzteil-Kommando [cite: 326]).
- [cite_start]**Live-Feedback**: Linien zwischen den Blöcken leuchten kurz auf, wenn ein Event über den `EventBus` fließt[cite: 85].

7.4 Session Replay & Analyse (History View)

[cite_start]Basierend auf dem `SessionManager`.

- **Konzept**: Eine Ansicht zum "Zurückspulen" vergangener Messungen.
- [cite_start]**Funktion**: Über eine Timeline kann eine aufgezeichnete Session (identifiziert durch die `session_id`) ausgewählt werden. Die UI zeigt dann die historischen Daten so an, als würden sie gerade live passieren.
- **Vergleichs-Modus**: Zwei Sessions können übereinandergelegt werden (z.B. Entladekurve von Batterie A vs. Batterie B).

7.5 Synoptic View (Prozessgrafik)

- **Konzept**: Ein statisches Hintergrundbild (z.B. ein Foto deines Versuchsaufbaus oder ein Schaltplan), auf dem die Live-Werte der Entitäten an den physikalisch korrekten Stellen eingeblendet werden.
- **Nutzen**: Extrem intuitive Überwachung von komplexen Verdrahtungen.

Sonstiges

Was ich mir sonst noch vorstellen könnte:

- Virtuelle Instrumente (Skins): Dass du für das UDP3305 eine View baust, die exakt so aussieht wie die Frontplatte des echten Geräts. Das macht die Bedienung im Web viel natürlicher.
- Webcam-Integration mit Overlay: Wenn dein Pi eine Kamera hat, könntest du den Videostream anzeigen und die Messwerte (z.B. Temperatur) direkt über das Bild legen (ähnlich wie Augmented Reality).
- Alarm-Management: Eine View, die nur dann aufpoppt, wenn Grenzwerte überschritten werden (z.B. BMS-Alarm).

7.6 Webcam & Augmented Reality (AR) Overlay

Diese View kombiniert visuelles Feedback der Hardware mit den Live-Daten des EventBus.

Architektur des Datenflusses

- **Video-Pfad**: Webcam → OpenCV → FastAPI StreamingResponse (MJPEG) → Browser <img>.
- **Daten-Pfad**: Hardware → EventBus → WebSocket → Browser Canvas.
- **Vorteil**: Die hohe Last des Videos beeinträchtigt nicht die Echtzeit-Messdaten auf dem Bus.

Features

- **AR-Overlay**: Positionierung von Messwerten direkt über dem Videobild (z.B. Temperaturanzeige direkt auf dem Kühlkörper im Bild).
- **Visual CV**: Optionale Bilderkennung im Backend, die Ergebnisse (z.B. “Gerät eingeschaltet”) als reguläre Samples auf den Bus publiziert.

Implementierung (Code-Skizze)

- **Backend**: Neuer API-Endpunkt unter `/api/video/stream`.
- **Frontend**: Dynamisches Canvas-Mapping. Koordinaten für Overlays werden in der `config.yaml` des Geräts gespeichert.

7.7 Visual Event Trigger (Virtual Sensor)

Zusätzlich zum Videostream kann das System Bildbereiche (ROI) analysieren, um “virtuelle Sensoren” zu generieren.

- **Funktion**: Überwachung von analogen Anzeigen oder LEDs, die keine Datenschnittstelle besitzen.
- **Verarbeitung**:

1. ROI Definition via Koordinaten.

```
  2. HSV-Farbraumfilterung zur Detektion von Statusfarben.
  3. State-Machine zur Vermeidung von Bus-Spam (nur Änderungen werden
publiziert).
* **Anwendung**: "BMS Alarm LED" -> EventBus -> "PSU OFF".
```

7.8 Optical Character Recognition (OCR) Sensor

Verwandelt visuelle Anzeigen in digitale Datenströme.

- **Technologie**: Integration von `Tesseract` oder SSOCR in den Webcam-Treiber.
- **Datenfluss**:

1. Extraktion der Anzeige via ROI.

```
  2. Bildvorbehandlung (Grayscale, Thresholding, Morphologie).
  3. Konvertierung String -> Float/Int.
  4. Publikation als ''NumericSample'' oder ''TextSample'' auf dem EventBus.
* **Anwendung**: Digitalisierung von Legacy-Hardware ohne Schnittstellen
(DMMs, Waagen, analoge Anzeigen).
```

8. Zusammenfassung der Datenfluss-Architektur

Der Datenfluss im erweiterten System folgt nun diesem Muster:

1. [cite_start]**Hardware/Input** (z.B. Owon HDS [cite: 270] [cite_start]oder Gamepad) → **Bus**[cite: 85].
2. [cite_start]**LogicService** (Abonniert Bus) → Berechnet Transformation → **Engine.execute_command**[cite: 84].
3. [cite_start]**Web-Views** (Abonnieren Bus via WebSocket [cite: 427]) → Filtern nach Focus-Lock → **Visualisierung** (3D, Table, Graph).

From:
https://drklipper.de/ - **Dr. Klipper Wiki**

Permanent link:
https://drklipper.de/doku.php?id=projekte:ionpy:ideen&rev=1771151696

Last update: **2026/02/15 11:34**