Projektdokumentation – Bluetooth Robotersteuerung

**Projektbeschreibung**

Dieses Projekt ermöglicht die Steuerung eines Roboters über Bluetooth mithilfe einer Kivy-App. Der Benutzer kann über Slider die Motoren steuern, Bluetooth-Geräte scannen und sich mit ihnen verbinden.

**Entwurfsentscheidungen**

* Kivy für GUI: Ermöglicht eine plattformübergreifende App mit einer dynamischen Benutzeroberfläche.
* Bleak für Bluetooth: Asynchrones Bluetooth-Management für eine nahtlose Verbindung und Steuerung.
* JSON für die Datenübertragung: Ermöglicht eine strukturierte und einfache Kommunikation mit dem Roboter.

**Klassendesign und Funktionen**

**Page1 (Startbildschirm der App)**

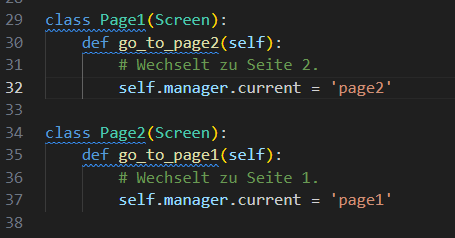
Diese Klasse erbt von Screen und stellt die erste Seite der App dar. Hier sind die Hauptfunktionen:

* go\_to\_page2(): Wechsel zur zweiten Seite der App.

**Page2 (Steuerungsseite für den Roboter)**

Diese Klasse erbt ebenfalls von Screen und enthält Bedienelemente zur Robotersteuerung.

* go\_to\_page1(): Wechsel zur ersten Seite.



**MyApp (Hauptklasse der App)**

Erbt von App (Kivy) und ist für das App-Fenster sowie alle Kernfunktionen verantwortlich.

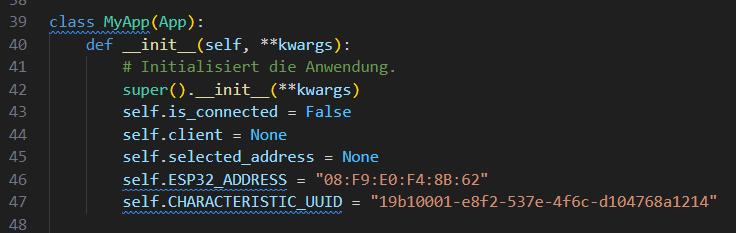
Designentscheidung: Alle Funktionen in MyApp zu zentralisieren, um die Implementierung und Wartung zu vereinfachen.

**Hauptfunktionen**

**def \_\_init\_\_(self)**

Initialisiert die App mit wichtigen Variablen:

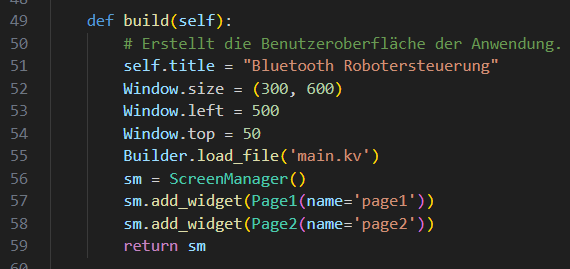
* self.is\_connected: Speichert den Verbindungsstatus.
* self.client: Referenz auf das verbundene Bluetooth-Gerät.
* self.selected\_address: Speichert die MAC-Adresse des ausgewählten Geräts.
* self.ESP32\_ADDRESS / self.CHARACTERISTIC\_UUID: Vordefinierte Werte für das ESP32-Gerät.



**def build(self)**

Erstellt die Benutzeroberfläche und initialisiert das ScreenManager-Objekt.

* Window.size, Window.left, Window.top: Setzt die Fenstergröße und Position für eine mobile Ansicht.
* Builder.load\_file("main.kv"): Lädt die Kivy-Layout-Datei.

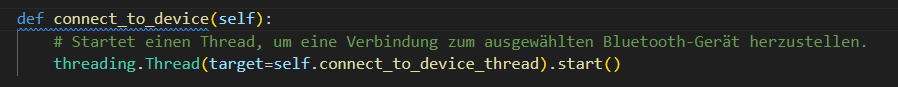
****

**Bluetooth-Funktionen**

**def connect\_to\_device(self)**

Startet einen separaten Thread zur Verbindung mit einem Bluetooth-Gerät.

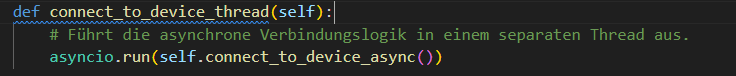
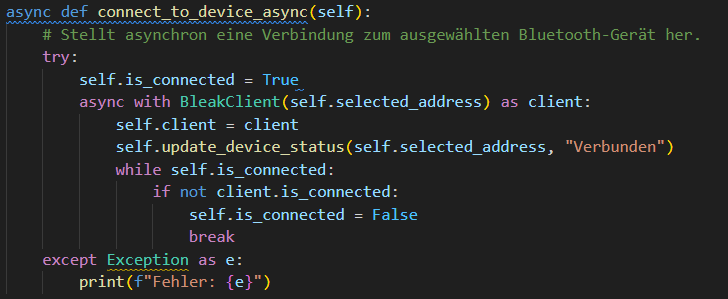
Warum ein Thread? Da BleakClient asynchron ist, verhindert ein separater Thread, dass die UI einfriert.



**async def connect\_to\_device\_async(self)**

Verbindet sich asynchron mit dem Bluetooth-Gerät.

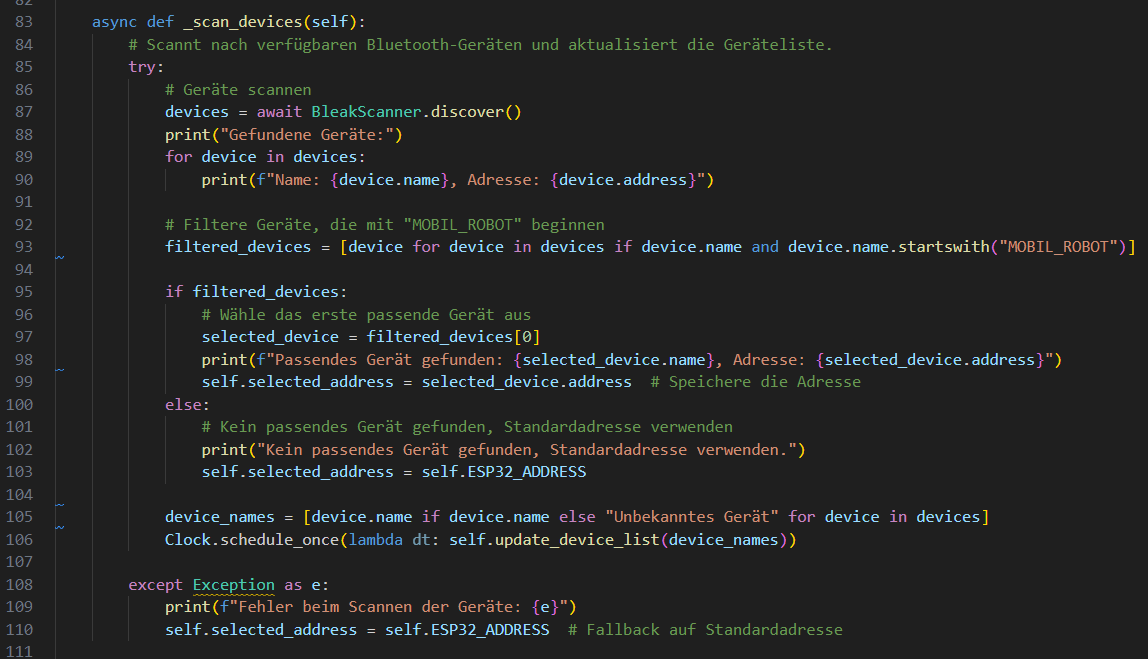
* await self.client.write\_gatt\_char(...): Sendet Daten an das Gerät.
* try-except: Fehlerhandling für Verbindungsprobleme.



**async def \_scan\_devices(self)**

Scannt nach Bluetooth-Geräten und aktualisiert die Liste.

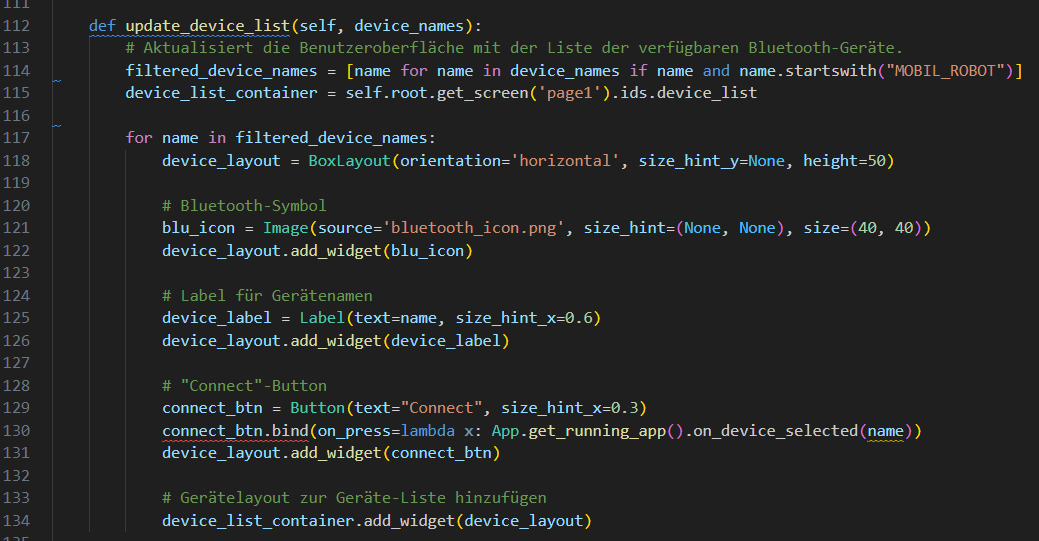
* BleakScanner.discover(): Findet alle erreichbaren Geräte.
* Clock.schedule\_once(lambda dt: self.update\_device\_list(device\_names)): Ruft die Aktualisierung der UI thread-sicher auf.



**def update\_device\_list(self, device\_names)**

Aktualisiert die Geräteliste in der UI.

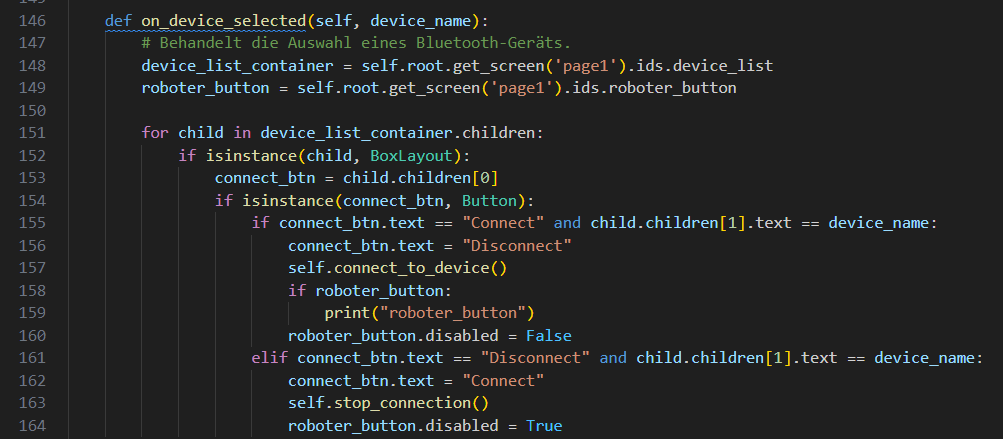
* Erstellt ein BoxLayout mit einem Bluetooth-Icon, einem Label und einem "Connect"-Button.
* Warum lambda?
  + lambda wird verwendet, um die Funktion mit Parametern zu binden, ohne dass sie sofort aufgerufen wird.



**def on\_device\_selected(self, device\_name)**

Reagiert auf einen Klick auf den "Connect"-Button.

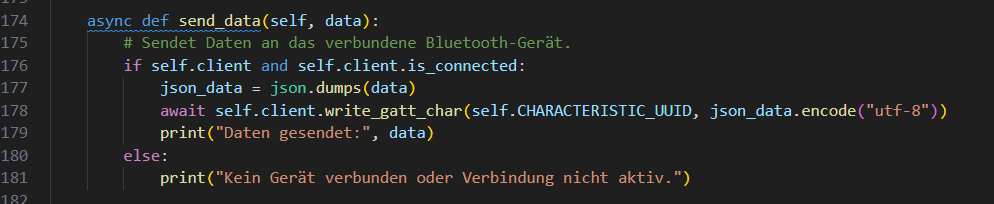
* Falls das Gerät bereits verbunden ist, wird die Verbindung getrennt.
* Falls nicht, wird connect\_to\_device() aufgerufen.



**async def send\_data(self, data)**

Sendet JSON-Daten an das verbundene Bluetooth-Gerät.

* Warum await?
  + await wird genutzt, um nicht-blockierend auf die Antwort des Geräts zu warten.

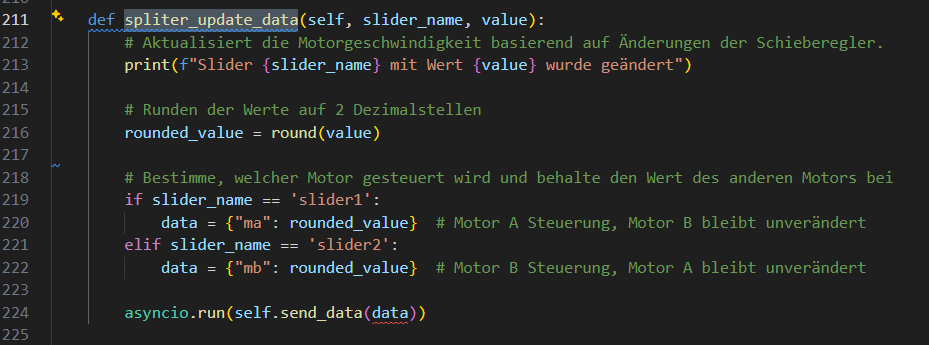
****

**Motorsteuerung**

**def spliter\_update\_data(self, slider\_name, value)**

Aktualisiert die Motorgeschwindigkeit, wenn ein Slider bewegt wird.

* Rundet die Werte und stellt sicher, dass sie im gültigen Bereich liegen.
* Sendet die aktualisierten Werte an den Roboter.

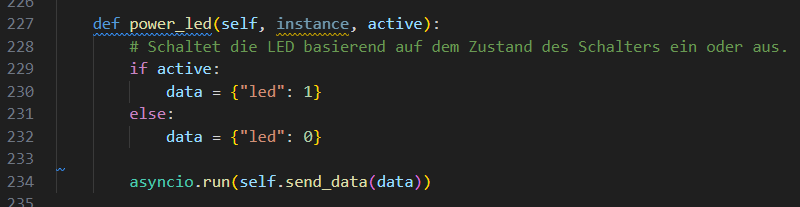
****

**LED-Steuerung**

**def power\_led(self, instance, active)**

Schaltet die LED des Roboters an oder aus.

* data = {"led": 1 if active else 0}
* Sendet die Daten an den Roboter per send\_data(data).



**Zusätzliche Erklärungen**

**Warum asyncio nutzen?**

* Asynchrone Funktionen verhindern das Blockieren der UI.
* asyncio erlaubt mehrere Tasks gleichzeitig (z. B. Scannen während der Steuerung).

**Warum Clock.schedule\_once() nutzen?**

* Kivy arbeitet nicht thread-sicher. Clock.schedule\_once() stellt sicher, dass UI-Updates im Hauptthread passieren.

**Warum threading.Thread() nutzen?**

* Manche asyncio-Aufrufe können nicht direkt aus der Haupt-Thread-UI gestartet werden.
* Threads verhindern, dass lange Prozesse die UI einfrieren.

**Qualitätssicherung**

* Tests: Manuelle Tests auf verschiedenen Geräten.
* Soll-Ist-Vergleich: Sicherstellen, dass die Steuerung flüssig funktioniert.
* Fehlerhandling: Nutzung von try-except, um Abstürze zu vermeiden.

**Erweiterung:**

Die Bluetooth-Robotersteuerung kann in Zukunft erweitert werden, sodass die Steuerung nicht nur über Buttons, sondern auch durch die Neigungssensoren eines Smartphones erfolgt.

**Quellen**

* **Bleak-Dokumentation:** [**https://bleak.readthedocs.io/**](https://bleak.readthedocs.io/)
* **Kivy-Dokumentation:** [**https://kivy.org/doc/stable/**](https://kivy.org/doc/stable/)
* **Python AsyncIO:** [**https://docs.python.org/3/library/asyncio.html**](https://docs.python.org/3/library/asyncio.html)