Einstieg in LoRaWAN mit dem Wio-E5 und Python

In diesem Workshop geben wir eine kurze Einführung in LoRaWAN und die Verwendung zusammen mit einem Seeedstudio Wio E5 und einem Raspberry Pi. Der verwendete Python Code sollte mit jedem Raspberry Pi funktionieren, wir verwenden für die Workshops der Einfachheit halber, einen Pi 400 oder Pi 500.

Was ist LoRaWAN?

The Things Network

The Things Network (TTN) versteht sich als globales, kollaboratives Ökosystem, das die LoRaWAN-Technologie durch einen offenen Community-Ansatz demokratisiert. Die besondere Bedeutung für die LoRaWAN-Community ergibt sich aus dem kostenlosen Zugang zu einer robusten Netzwerk-Infrastruktur, die es Bastlern und Profis gleichermaßen erlaubt, Sensordaten über große Distanzen zu übertragen, ohne auf teure kommerzielle Anbieter angewiesen zu sein. Die Integration erfolgt dabei flexibel über den TTN-Stack, der Daten via MQTT oder Webhooks an Endanwendungen weiterreicht und durch integrierte Payload-Formatierer die Interpretation binärer Datenströme vereinfacht. Da TTN auf dem Crowdsourcing-Prinzip basiert – jeder kann ein Gateway beisteuern und so die Abdeckung erweitern – wächst das Netzwerk organisch durch seine weltweiten Communities und schafft so eine flächendeckende Konnektivität, die auf gegenseitiger Unterstützung und offenen Standards fußt.

Hier geht es zur Registrierung: Jetzt bei The Things Network registrieren Registeriere dich, und lege deine erste Application an, hier fügen wir dann später unseren Wio-E5 ein.

Raspberry Pi und E5 verbinden

Verbinde den Raspberry Pi und den E5 über die Pins des Raspberry Pi´s wie folgt:

- 3.3V → Rotes Kabel
- Ground → Schwarzes Kabel
- UART0 Tx (GPIO 14) → Weißes Kabel
- UART0 Rx (GPIO 15) → Gelbes Kabel

[Suche gerne nach einem sogenannten Pinout im Internet zu dem von dir verwendeten Pi damit du die Kabel richtig anschließen kannst]

Die Kabelfarben stimmen mit den offiziellen Grove Kabel Farben überein. 3.3V und Ground sind für die Spannungsversorgung zuständig. Tx und Rx sind Teil des von dem Wio-E5 Modul verwendeten Serial Protokolls. Tx steht für Transmit und Rx für Receive. Hier ist Obacht geboten! Dies ist dann keine 1:1 Verbindung sondern diese müssen Überkreuz angeschlossen werden. Rx vom Modul geht zum Tx vom

Raspberry Pi und so auch das Tx vom Modul zum Rx vom Pi.

Um die serielle Schnittstelle auf dem Raspberry Pi zu aktivieren, nutzt man den Befehl `sudo raspi-config` welchen man im Terminal eingibt (öffnen kann man dies über den schwarzen Kasten in der oberen Leiste). Navigiere dort zu **Interface Options** → **Serial Port** (Navigation über Pfeiltasten und Enter). Deaktiviere die **Login shell** (No) und aktiviere die **Serial port hardware** (Yes). Ein anschließender Neustart mit `sudo reboot` ist erforderlich.

Das Terminal werden wir auch im weiteren Teil des Workshops verwenden.

Was sind AT-Befehle und wie funktionieren sie?

AT-Befehle (abgeleitet von **AT**tention) sind ein standardisierter Befehlssatz zur Steuerung von Modems und Funkmodulen über eine serielle Schnittstelle (UART). In diesem Workshop dienen sie als Brücke zwischen dem Raspberry Pi und dem LoRa-E5 Modul, um komplexe LoRaWAN-Prozesse durch einfache Textbefehle zu steuern.

Funktionsweise im Projekt

Die Kommunikation erfolgt nach einem einfachen Frage-Antwort-Prinzip: Der Raspberry Pi sendet eine Zeichenfolge an das Modul, und das Modul führt die entsprechende Aktion aus oder gibt Informationen zurück.

- **Syntax:** Jeder Befehl beginnt mit dem Präfix AT. Parameter werden meist mit einem Gleichheitszeichen (=) angehängt, während Abfragen oft durch ein Fragezeichen oder den reinen Befehl erfolgen.
- **Testen:** Ein einfaches AT dient als Verbindungstest; das Modul antwortet im Idealfall mit “OK”.
- **Konfiguration:** Befehle wie `AT+ID` werden genutzt, um Hardware-spezifische Adressen wie die DevEui oder AppEui auszulesen, die für die Registrierung im Netzwerk zwingend erforderlich sind.
- **Aktion:** Mit Befehlen wie `AT+JOIN` wird der komplexe Prozess der Netzwerkanmeldung gestartet. Erst nach einer erfolgreichen Bestätigung durch das Modul (“Network joined”) können weitere Befehle zum Senden von Daten (z. B. `AT+MSG` für Text oder `AT+MSGHEX` für Hexadezimalwerte) genutzt werden.

Tests mit Python (Beispiel Skript)

Für eine komfortable Steuerung des LoRa-Moduls steht das Skript `LoRaWorkshop.py` zur Verfügung. Dieses automatisiert die AT-Befehle und bietet eine menügeführte Oberfläche im Terminal.

Vorbereitung

Bevor das Skript gestartet werden kann, muss die benötigte Python-Bibliothek installiert werden:

```
pip install pyserial
```

Hier ist außerdem das Skript. Referenziere es gerne um bestimmte Funktionen zu verstehen:

LoRaWorkshop.py

```
import serial
import time

PORT = "/dev/serial0"
BAUD = 9600

def send_command(ser, cmd, wait_for="OK", timeout=5):
    """Sendet einen Befehl und wartet auf eine Antwort."""
    ser.write(f"{cmd}\r\n".encode()) # wichtig ist \r\n zur bestätigung
im Emulator
    start_time = time.time()
    response = ""

    while (time.time() - start_time) < timeout:
        if ser.in_waiting > 0:
            line = ser.readline().decode(errors='ignore').strip()
            if line:
                print(f"  [Modul]: {line}")
                response += line
                if wait_for in line:
                    return True
    return False

def main_menu():
    try:
        ser = serial.Serial(PORT, BAUD, timeout=1)
        print("--- LoRa-E5 Schnellwahl-Terminal ---")

        while True:
            print("\nWas küt ?")
            print("1 Test (AT)")
            print("2 IDs auslesen (AT+ID)")
            print("3 Mit TTN verbinden (AT+JOIN)")
            print("4 Nachricht senden (Text)")
            print("5 Nachricht senden (Hex)")
            print("6 Reset des Moduls") # Falls timeout wegen zuviel
anfragen
            print("7 Nachricht im Loop senden (alle 45s)")
            print("c Beliebigen AT-Befehl senden")
            print("q Beenden")
            # print("f Mit TTN verbindung Forcen (AT+JOIN=FORCE)")

            choice = input("\nAuswahl: ").strip().lower()

            if choice == '1':
```

```
                send_command(ser, "AT")

            elif choice == '2':
                send_command(ser, "AT+ID")

            elif choice == '3':
                print("Join-Prozess gestartet...")
                keywords = ["done", "joined already"]
                ser.write(b"AT+JOIN\r\n")
                start_time = time.time()
                joined = False

                while (time.time() - start_time) < 30:
                    if ser.in_waiting > 0:
                        line =
ser.readline().decode(errors='ignore').strip()
                        if line:
                            print(f"  [Modul]: {line}")
                            line_lower = line.lower()
                            if any(key in line_lower for key in
keywords):
                                joined = True
                                break
                            if "failed" in line_lower or "busy" in
line_lower:
                                break

                if joined:
                    print("Bereit: Modul ist im Netz.")
                else:
                    print("Timeout oder Fehler beim Join.")

            elif choice == '4':
                msg = input("Gib deine Nachricht ein: ")
                send_command(ser, f'AT+MSG="{msg}"')

            elif choice == '5':
                hex_payload = input("Gib den Hex-Code ein (z.B. AA BB
01): ")
                hex_payload = hex_payload.replace(" ", "") # falls
Eingabe Leerzeichen beinhaltet
                send_command(ser, f'AT+MSGHEX="{hex_payload}"')

            elif choice == '6':
                send_command(ser, "AT+RESET")

            elif choice == 'f':
                send_command(ser, "AT+JOIN=FORCE")

            elif choice == '7':
                msg = input("Gib die Nachricht ein, die geloopt werden
```

```
soll: ")
                print("\nLoop gestartet. Drücke STRG+C (bzw. Ctrl+C),
um den Loop zu beenden!")
                try:
                    while True:
                        print(f"\nSende Nachricht: {msg}")
                        send_command(ser, f'AT+MSG="{msg}"') # Auch mit
Sensordaten Möglich einfach msg ändern zu dem Sensor Output
                        print("Warte 45 Sekunden auf die nächste
Sendung...")
                        time.sleep(45)  # Zeitintervall zur loop
übermittlung
                                        # Es gibt Gesetzliche Vorgaben
(5min) für den Abstand von Signalen
                                        # Wollen ja nicht die
Bundesnetzagentur im nacken haben
                except KeyboardInterrupt:
                    print("\nLoop durch Benutzer abgebrochen. Kehre zum
Menü zurück...")

            elif choice == 'c':
                msg = input("command: ")
                send_command(ser, msg)

            elif choice == 'q':
                print("Programm beendet.")
                ser.close()
                break
            else:
                print("Ungültige Auswahl.")

            time.sleep(1)

    except serial.SerialException as e:
        print(f"Fehler: Konnte Port {PORT} nicht öffnen ({e})")

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

Skript starten

Navigiere im Terminal in den Ordner, in dem die Datei liegt, und führe sie aus:

```
python LoRaWorkshop.py
```

Funktionsübersicht

Nach dem Start erscheint ein Auswahlmenü mit folgenden Optionen:

Auswahl	Funktion	Beschreibung
1	Test (AT)	Prüft, ob die RX/TX-Verbindung zum Modul steht.
2	IDs auslesen	Zeigt DevEui, DevAddr und AppEui für die TTN-Registrierung an.
3	Mit TTN verbinden	Startet den Join-Prozess (AT+JOIN). Ein Gateway muss erreichbar sein.
4	Nachricht senden (Text)	Übermittelt eine Nachricht im Klartext.
5	Nachricht senden (Hex)	Übermittelt Daten als Hexadezimal-Zeichen (ASCII-Code).
6	Reset	Startet das LoRa-Modul neu, falls es nicht reagiert.
7	Loop-Modus	Sendet automatisch alle 45 Sekunden eine Nachricht.
f	Verbindung erzwingen	Verlässt bestehende Sitzungen und erzwingt einen neuen Join (Force).
q	Beenden	Schließt das Skript (auch per STRG+C möglich).

> **Hinweis:** Die Funktionen zur Datenübertragung (Text/Hex/Loop) stehen erst zur Verfügung, wenn die Netzwerkanfrage erfolgreich war und das Modul “Network joined” meldet.

Payload Decoder

Um die Daten, welche in TTN über LoRaWAN ankommen, zu decodieren kann ein sogenannter Payload Decoder hinterlegt werden. Da die Daten in diesem Fall über die ASCII Tabelle (Gerne mal googeln) codiert wurden in Hexadezimal, sieht der Decoder in diesem Fall so aus:

decoder.js

```
function decodeUplink(input) {
  var bytes = input.bytes;

  // Konvertierung der Bytes in einen String (ASCII)
  var text = "";
  for (var i = 0; i < bytes.length; i++) {
    text += String.fromCharCode(bytes[i]);
  }

  return {
    data: {
      type: "TEXT_MSG",
      message: text,
      length: bytes.length
    }
  };
}
```

Credits

Die Entwicklung des Python Scripts sowie die Anleitung für die manuellen Tests wurden durch den

Studenten Tim Oswald im Rahmen seiner Bachelorarbeit durchgeführt. Hier der Link zu seinem erstellten Cheatsheet:

Cheatsheet