**Installation och körning**

1. Skapa en **virtuell Python-miljö** och aktivera den (t.ex. kör python3 -m venv venv och sedan source venv/bin/activate på Linux/Mac eller motsvarande på Windows).
2. Installera projektets beroenden. Gå till projektets rotkatalog och kör pip install -e . för att installera CANopen-biblioteket (motot\_config) lokalt. Installera dessutom nödvändiga paket för exempelprogrammen, såsom python-can, bitstring, PySide6, numpy, scipy, matplotlib och nidaqmx.
3. Se till att din CAN-hårdvara (t.ex. en Kvaser CAN-anslutning) och eventuella andra enheter (motor, DAQ) är anslutna och redo att användas.
4. Starta huvudprogrammet (GUI) genom att köra filen gui.py i exempelkatalogen. Exempelvis: python examples/gui.py. Det grafiska gränssnittet öppnas därefter.

**Projektstruktur**

Projektets rotkatalog innehåller bland annat följande kataloger och filer:

* **canopen/** – Själva CANopen-biblioteket i Python. Här finns moduler för CANopen-nätverkshantering (NMT), PDO/SDO-kommunikation, synkronisering och andra grundläggande funktioner.
* **examples/** – Katalogen med användarens egna exempelkod och data för motorstyrning, sensormätning och visualisering.
* **examples/help\_programs/** – Hjälpprogram som utför specifika uppgifter (t.ex. konfiguration, fabriksåterställning, dataloggning) för CANopen-noder och hårdvara.
* **doc/** – Dokumentation (Sphinx) för CANopen-biblioteket och projektexemplen. Innehåller texter om protokollet, bibliotekets funktioner och exempel på användning.
* **test/** – Testkod (PyTest) för att verifiera att biblioteket fungerar som förväntat.

**Exempelfiler i examples/**

* **gui.py** – Huvudprogrammet med grafiskt användargränssnitt (GUI). Här initieras drivkretsarna och trådar för datainsamling startas. GUI:t hanterar användarinmatning för att styra motorerna och visa mätdata i realtid.
* **motor\_control.py** – Innehåller funktioner för direkt styrning av motor via CANopen. Här definieras funktioner för att initiera CANopen-noden, ändra driftsläge, sätta målposition (position och vridmoment), läsa av enkodersignal och hantera andra parametrar.
* **plot\_test\_data.py** – Program för att läsa in och plotta testdata från CSV-filer. Skriptet läser sparad testdata (t.ex. position och kraft) och genererar diagram med matplotlib för att underlätta analys.
* **dac.py** – Skript för att hantera DAQ-enhetens in- och utgångar. Det innehåller funktioner för att mäta belastningsceller och generera styrsignaler via National Instruments hårdvara.
* **qe\_command.py** – Hanterar CAN-trafik och kvadraturavkodning. Funktionen qe\_read läser CAN-meddelanden (PGN) från **qe\_command**, beräknar rörelseriktning och uppdaterar de värden som GUI:t kan visa i realtid.
* **set\_pos\_torque.py** – Exempelprogram för position- och vridmomentstyrning. Innehåller rutiner för att växla driftsläge mellan position och vridmoment, enkel homing-sekvens, samt funktioner för att spara testdata och visa eventuella felkoder från drivern.
* **worker\_pool.py** – Enkel trådpool-implementation. Används för att köra bakgrundsuppgifter parallellt (t.ex. kontinuerlig datainsamling) utan att blockera huvudprogrammet.

**Hjälpprogram (examples/help\_programs/)**

* **Auto\_setup.py** – Skript som automatiskt ansluter till CANopen-nätverket och laddar in drivrutinskonfiguration. Underlättar initialinställningen av nod-ID och nätverksparametrar.
* **config\_pdo.py** – Konfigurerar PDO-mappning för CANopen-noder. Exempel på hur man sätter upp önskade SDO/PDO-inställningar vid uppstart.
* **dac\_input\_plot.py** – Visualiserar realtidsdata från DAQ-ingångar. Används för att animera och visa insignaler (t.ex. från sensorer eller styrkretsar) med matplotlib.
* **factory\_reset.py** – Utför fabriksåterställning av en Nanotec-motorstyrning (t.ex. PD4-E). Återställer alla parametrar till ursprungsläget via CANopen-kommandon.
* **fft\_test.py** – Utför en FFT-analys (Fourier-transform) på insamlade signaler. Skriptet läser in data från DAQ och utför frekvensanalys för att testa motor- eller sensoregenskaper.
* **save\_data.py** – Hjälpmodul för att skapa och spara testdatafiler (CSV). Skriver tidsstämplar och mätvärden till fil för senare analys.
* **test\_offcet.py** – Testprogram för att justera offset på mätsignaler. Används för att kalibrera lastcell eller kontrollera referensvärden.
* **test\_pdo\_maping.py** – Verifierar PDO-mappning och kommunikation på CANopen-noden. Skriptet testar att in- och utdata mappas korrekt och fungerar enligt förväntan.