Day 5 - Session 3:
VISA Instrument Communication

This notebook introduces **VISA (Virtual Instrument Software Architecture)** communication in Python using the pyvisa library. VISA is a widely adopted standard that allows you to control many different types of laboratory instruments (like oscilloscopes, multimeters, power supplies, function generators) regardless of their connection interface (USB, GPIB, Ethernet/LAN, RS-232 serial). This is a crucial skill for automating experiments, collecting data, and managing test setups in engineering and scientific research.

1. Introduction to VISA and pyvisa

Imagine you have many different lab instruments, and each one uses a different cable (USB, Ethernet, GPIB) and speaks a slightly different "language" (command set). It would be very hard to write a single program to control all of them.

- **VISA (Virtual Instrument Software Architecture):** This is a standard that creates a unified way to communicate with instruments, regardless of their underlying connection type or specific manufacturer. It acts as a "universal translator" and "universal connector."
- **pyvisa:** This is a Python library that provides a user-friendly interface to the VISA standard. It allows your Python code to talk to any VISA-compliant instrument.

Why Use VISA and pyvisa?

- **Unified Interface:** Control different instruments (oscilloscopes, DMMs) from various manufacturers through a single programming approach.
- **Interface Agnostic:** Works over USB, GPIB, Ethernet, RS-232 serial – you write code once, and VISA handles the low-level communication specifics.
- **Automation:** Automate repetitive measurements, experiments, and test sequences, saving time and reducing human error.
- **Data Acquisition:** Programmatically collect large datasets from instruments.

Prerequisites for pyvisa:

To use pyvisa, you need:

1. **Python:** (Already have this!)
2. **pyvisa library:** Install it using pip install pyvisa.
3. **A VISA Backend (Library):** This is the actual software that implements the VISA standard on your computer. Common ones include:
    - **NI-VISA (National Instruments VISA):** Very common and robust. (Requires installation, e.g., from NI website).
    - **Keysight VISA (Keysight Technologies):** Another popular option.
    - **PyVISA-py:** A pure Python backend included with pyvisa. It works without needing NI-VISA, but it's often slower and might not support all features or instruments as reliably as a full vendor-supplied VISA library. It's great for getting started!

For this notebook, we will use PyVISA-py by default, so you don't need to install extra vendor software to follow along conceptually. However, for real lab work, installing NI-VISA or Keysight VISA is usually recommended.

2. Setting Up pyvisa and Finding Resources

The first step is to tell pyvisa where to find your instruments.

2.1 Importing pyvisa and Creating a Resource Manager

Conceptual Approach:

pyvisa needs a "Resource Manager" to discover and manage connections to instruments. This is like the central directory for all your lab equipment.

pyvisa.ResourceManager(): How to Get a Resource Manager

- **Purpose:** Creates an object that finds and manages VISA resources.
- **Syntax:** rm = pyvisa.ResourceManager(visa_library=None)
    - visa_library: (Optional) Path to your VISA backend DLL (e.g., 'C:\\Windows\\System32\\visa32.dll' for NI-VISA). If not provided, pyvisa tries to find one or uses PyVISA-py (default for pyvisa).

Python

```
# ----------------------------------------
# VISA Communication Setup using PyVISA
# ----------------------------------------

import pyvisa  # PyVISA is used to talk to instruments like oscilloscopes, multimeters, etc.

print("--- Setting Up pyvisa ---")

try:
    # Create a VISA Resource Manager (used to list and connect to instruments)
    # By default, PyVISA uses National Instruments (NI-VISA) backend.
    # '@py' forces the use of PyVISA-py (pure Python backend, no need for NI software)

    rm = pyvisa.ResourceManager('@py')  # '@py' is best for testing without actual hardware
    print("✅ PyVISA Resource Manager created successfully.")
    print(f"🔌 Backend in use: {rm.backend}")

except Exception as e:
    print("❌ Error while creating Resource Manager:")
    print(f"   → {e}")
    print("💡 Tip: Make sure 'pyvisa' is installed correctly.")
    print("         If you're using hardware with NI-VISA, ensure NI-VISA drivers are also installed.")
    print("📘 Fallback: Continuing with conceptual examples only.")

    rm = None  # If setup fails, set rm to None to avoid crashes in further code
```

2.2 Listing Available Resources (rm.list_resources())

Once you have a Resource Manager, you can ask it to list all the instruments it can see connected to your computer.

rm.list_resources(): How to Find Instruments

- **Purpose**: Discovers and lists the addresses (names) of all available VISA instruments.
- **Syntax**: resource_list = rm.list_resources()
- **Output**: Returns a tuple of strings, where each string is a unique VISA address.

Common VISA Instrument Address Formats:

Interface	Example Address	Description
USB	USB0::0xXXXX::0xYYYY::SN1234::INSTR	USB instrument with Vendor ID (XXXX), Product ID (YYYY), Serial Number (SN1234).
GPIB	GPIB0::1::INSTR	GPIB bus 0, device address 1.
TCPIP	TCPIP0::192.168.1.100::INSTR or TCPIP0::myinst::inst0::INSTR	Network instrument at IP address 192.168.1.100 or hostname myinst.
ASRL	ASRL1::INSTR or ASRL1::9600::8N1::INSTR	Asynchronous Serial (COM port) on COM1 (Windows) or /dev/ttyS0 (Linux). 9600 is baud rate.

```python
# File: list_visa_resources.py
```

```python
import pyvisa  # PyVISA is a Python library to communicate with instruments via VISA

print("\n🔍 --- VISA Instrument Discovery ---")

try:
    # Create a VISA Resource Manager
    # '@py' uses the PyVISA-py backend (pure Python, no NI-VISA needed)
    rm = pyvisa.ResourceManager('@py')
    print("✅ PyVISA Resource Manager created successfully.")
    print(f"Backend in use: {rm.backend}")

    try:
        # List all available VISA resources
        resources = rm.list_resources()
        print("\n📡 Found the following VISA resources:")

        if resources:
            for res in resources:
                print(f"  • {res}")
        else:
            print("⚠️ No VISA instruments found.")
            print("ℹ️  This is normal if no physical instruments are connected.")
            print("💡 You can test with TCPIP0::IP::5025::SOCKET for a Raspberry Pi SCPI server.")

    except pyvisa.errors.VisaIOError as e:
        print(f"❌ Error listing VISA resources: {e}")
        print("🔧 This may happen if the VISA backend is not correctly installed or configured.")

except Exception as e:
    print(f"❌ Failed to create VISA Resource Manager: {e}")
    print("🔧 Ensure 'pyvisa' is installed. If using NI-VISA, install its driver separately.")
```

3. Opening a Connection (rm.open_resource())

Once you know the address of an instrument, you can open a connection to it.

3.1 rm.open_resource(): Connecting to an Instrument

Conceptual Approach:

You use the Resource Manager to "open" a specific instrument resource by its address. This gives you an "instrument object" which you'll use to send commands and read data.

rm.open_resource(): How to Open a Connection

- **Purpose:** Opens a connection to a specific VISA instrument.
- **Syntax:** inst = rm.open_resource(resource_name, ...)
    - resource_name: The VISA address string (e.g., 'USB0::0xXXXX::INSTR').
- **Output:** Returns an Instrument object.

Important: Always Close Connections!

Just like files, you should always close your instrument connection when you're done. The with statement (as a context manager) is the best way to do this, as it guarantees the connection is closed even if errors occur.

Python

```
# File: open_visa_instrument.py

import pyvisa  # Import the PyVISA library

print("\n--- Opening a Connection to an Instrument ---")

# Step 1: Create a Resource Manager using PyVISA-py backend
try:
    rm = pyvisa.ResourceManager('@py')  # '@py' ensures using the pure Python
backend (no NI-VISA needed)
    print("✅ PyVISA Resource Manager created successfully.")
    print(f"Backend: {rm.backend}")
```

```
except Exception as e:
    print(f"❌ Failed to create Resource Manager: {e}")
    rm = None

# Step 2: Define a dummy VISA address (or use one from rm.list_resources())
dummy_instrument_address = 'ASRL1::INSTR'  # Example: Serial port or test
resource

# Step 3: Try to open a connection to the resource
if rm:
    try:
        # 'with' block automatically closes the connection when done
        with rm.open_resource(dummy_instrument_address) as my_instrument:
            print(f"\n🔗 Successfully connected to:
{dummy_instrument_address}")
            print(f"Interface Type: {my_instrument.interface_type}")  # e.g.,
'ASRL', 'GPIB', 'USB'
            print(f"Resource Name: {my_instrument.resource_name}")
            print("📡 Connection is active inside this 'with' block.")
            # You could send SCPI commands here, e.g.,
my_instrument.query("*IDN?")

        # Outside the 'with' block, the connection is automatically closed
        print(f"🔒 Connection to {dummy_instrument_address} is now closed.")

    except pyvisa.errors.VisaIOError as e:
        print(f"\n⚠️ Could not open resource {dummy_instrument_address}: {e}")
        print("ℹ️ This might happen if no instrument is connected or address is
incorrect.")
    except Exception as e:
        print(f"❌ Unexpected error: {e}")
else:
    print("❌ Resource Manager not available. Cannot open instrument.")
```

4. SCPI Commands: The Language of Instruments

Most modern programmable instruments understand **SCPI (Standard Commands for Programmable Instruments)**. SCPI defines a common set of commands used to control instruments and query their status.

4.1 SCPI Command Structure

SCPI commands look like text strings. They are hierarchical, often starting with a subsystem, followed by a command, and then parameters.

General Structure:

SUBSYSTEM:COMMAND [PARAMETER]; SUBSYSTEM:COMMAND?

- **Keywords:** Commands are typically uppercase.
- **Shorthand:** Many commands have a full (long) form and a short form (e.g., MEASURE:VOLTAGE:DC? can be MEAS:VOLT:DC?).
- **Parameters:** Values passed to the command (e.g., VOLT 5.0 for 5 volts).
- **Query (?):** Appending a ? to a command asks the instrument for its current setting or a measurement.
- **Separators:**
  - : separates levels in the hierarchy (e.g., MEASURE:VOLTAGE).
  - (space) separates a command from its parameters.
  - ; separates multiple commands on the same line.

Table: SCPI Command Examples

SCPI Command	Purpose	Query Form
`*IDN?`	Identity Query: Ask the instrument to identify itself (manufacturer, model, serial, firmware). This is almost always the first command you send.	`*IDN?`
`MEASURE:VOLTAGE:DC?`	Measure DC voltage.	`MEAS:VOLT:DC?`
`SOURCE:VOLTAGE:LEVEL 5.0`	Set source voltage level to 5.0 units.	`SOUR:VOLT:LEV?`
`TRIGGER:SOURCE IMM`	Set trigger source to immediate.	`TRIG:SOUR?`

`READ?`	Read data from the instrument (often combined with a trigger).	`READ?`

4.2 Writing and Reading Data: inst.write(), inst.read(), inst.query()

The Instrument object from pyvisa provides methods to send SCPI commands and receive responses.

Conceptual Approach:

- inst.write(): Sends a command to the instrument (no response expected back, like setting a parameter).
- inst.read(): Reads a response from the instrument (used *after* sending a query command).
- inst.query(): Combines write() and read() into one convenient method (sends a command with ? and waits for a response). This is commonly used.

Python

```python
import pyvisa

print("\n--- SCPI Commands: Writing and Reading Data ---")

# Step 1: Create a PyVISA Resource Manager (if not already created)
try:
    rm = pyvisa.ResourceManager('@py')  # '@py' uses the PyVISA-py backend
    print("✅ PyVISA Resource Manager created.")
except Exception as e:
    print(f"❌ Failed to create Resource Manager: {e}")
    rm = None

# Step 2: Interact with a simulated SCPI instrument
if rm:
    try:
        # Simulated instrument address (works only if the simulator backend
supports it)
        simulated_instrument_address = 'ASRL::SIM::INSTR'  # You can also try
'TCPIP::127.0.0.1::INSTR'
```

```
        with rm.open_resource(simulated_instrument_address) as inst:
            print(f"\n🔗 Connected to simulated instrument:
{inst.resource_name}")

            # --- 1. Query identity ---
            idn_response = inst.query('*IDN?')  # Standard SCPI command for
instrument identity
            print(f"🆔 Instrument ID: {idn_response.strip()}")

            # --- 2. Set a frequency using a sine wave command ---
            frequency_to_set = 1000  # Hz
            inst.write(f'APPL:SIN {frequency_to_set}')  # SCPI command to apply
sine waveform
            print(f"📡 Sent command: APPL:SIN {frequency_to_set}")

            # --- 3. Query the current frequency ---
            freq_response = inst.query('APPL:SIN?')
            print(f"🔍 Queried Frequency: {freq_response.strip()} Hz")

            # --- 4. Measure a voltage value ---
            voltage_response = inst.query('MEAS:VOLT:DC?')  # SCPI command to
measure DC voltage
            print(f"🔋 Measured DC Voltage: {voltage_response.strip()} V")

            # --- 5. Send multiple SCPI commands in one line ---
            inst.write('VOLT 2.5; OUTP ON')  # Set voltage and enable output
            print("⚙️ Sent: VOLT 2.5; OUTP ON")

            # --- 6. Query output state after write ---
            inst.write('OUTP?')       # Ask if output is ON or OFF
            output_state = inst.read()  # Read the result
            print(f"✅ Output State: {output_state.strip()}")

    except pyvisa.errors.VisaIOError as e:
        print(f"\n⚠️ VISA I/O Error: {e}")
        print("💡 This is common if no simulation backend is active or address
is invalid.")
    except Exception as e:
        print(f"\n❌ Unexpected error: {e}")
else:
    print("❌ No Resource Manager available. Cannot proceed.")
```

5. Reading Data and Setting Configurations (Putting it Together)

This section provides a more complete example of how you would interact with an instrument to set up a measurement and then take readings.

Conceptual Approach:

1. Connect to the instrument.
2. Set its measurement mode/range (configuration).
3. Trigger a measurement or read a value.
4. Optionally, change another setting and repeat.
5. Close the connection.

Simulation over RaspberryPi (As a VISA Supported over TCP/IP)

Python

```python
import socket
import subprocess
import random
import RPi.GPIO as GPIO

# --- GPIO Optional Setup (if needed) ---
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# --- Function: Read CPU temperature ---
def get_rpi_temperature():
    try:
        output = subprocess.check_output(['vcgencmd', 'measure_temp']).decode()
        temp_str = output.strip().split('=')[1].replace("'C", "")
        return float(temp_str)
    except Exception as e:
        print(f"Error reading temperature: {e}")
        return -1.0

# --- Function: Simulate Resistance Reading ---
def simulate_resistance():
    return round(random.uniform(500.0, 600.0), 2)  # e.g., 500–600 Ohms
```

```
# --- SCPI TCP Server ---
HOST = '0.0.0.0'
PORT = 5025

print(f"🔌 SCPI Server starting on port {PORT}...")

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_socket.bind((HOST, PORT))
    server_socket.listen(5)
    print("🟢 Waiting for SCPI client connections...")

    while True:
        conn, addr = server_socket.accept()
        print(f"✅ Connected from: {addr}")

        with conn:
            data = conn.recv(1024)
            if not data:
                continue

            command = data.decode().strip().upper()
            print(f"📥 Received: {command}")

            # --- Process SCPI Commands ---
            if command == "*IDN?":
                response = "RPi,SCPI-GPIOSIM,001,1.0\n"
            elif command == "MEAS:TEMP?":
                temperature = get_rpi_temperature()
                response = f"{temperature:.2f}\n"
            elif command == "MEAS:RES?":
                resistance = simulate_resistance()
                response = f"{resistance:.2f}\n"
            else:
                response = "ERR:UNKNOWN_COMMAND\n"

            # Send response
            conn.sendall(response.encode('utf-8'))
            print(f"📤 Sent: {response.strip()}")
```

Python

```python
import socket
import time
import random

# Raspberry Pi SCPI Client for Voltage and Resistance simulation

def query_rpi_scpi(command, host="192.168.0.103", port=5025):
    """
    Sends a SCPI command to the Raspberry Pi SCPI server and returns the
response.
    """
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
        client_socket.connect((host, port))
        client_socket.sendall((command + '\n').encode('utf-8'))
        response = client_socket.recv(1024).decode('utf-8').strip()
        return response

# Simulate multiple voltage readings
print("\n--- Simulating DC Voltage Measurements from Raspberry Pi ---")

try:
    # Identify the instrument
    idn = query_rpi_scpi("*IDN?")
    print(f"ID: {idn}")

    # Simulate configuration step (for conceptual flow)
    print("Configured Raspberry Pi for DC Voltage measurement.")

    # Take multiple readings (simulate noise in values)
    num_readings = 5
    readings = []
    for i in range(num_readings):
        temp_reading = query_rpi_scpi("MEAS:TEMP?")
        try:
            temp_val = float(temp_reading) + random.uniform(-0.01, 0.01)  # add
noise
            readings.append(temp_val)
            print(f"  Reading {i+1}: {temp_val:.2f} °C")
        except ValueError:
            print(f"  Reading {i+1}: Invalid value received: '{temp_reading}'")
        time.sleep(0.5)
```

```
    # Display summary
    if readings:
        average = sum(readings) / len(readings)
        print(f"\nAverage Temperature: {average:.2f} °C")

    # Simulate resistance measurement
    print("\nNow simulating a resistance measurement...")
    resistance_sim = round(random.uniform(500.0, 600.0), 2)
    print(f"Measured Resistance (simulated): {resistance_sim} Ohms")

except Exception as e:
    print(f"Error: {e}")
```

6. Key Takeaways

This session provided a practical introduction to communicating with laboratory instruments using the VISA standard and pyvisa, a critical skill for test and measurement automation.

- **VISA & pyvisa:** Understood VISA as the universal standard for instrument communication and pyvisa as the Python library for implementing it.
- **Resource Management:** Learned to create a ResourceManager to discover available instruments (rm.list_resources()) and open connections to them (rm.open_resource()).
- **Safe Connections:** Emphasized the importance of using the with statement for pyvisa connections to ensure instruments are properly closed.
- **SCPI Commands:** Gained familiarity with SCPI as the common command language for instruments, understanding its hierarchical structure and the use of queries (?).
- **Instrument Communication Methods:** Mastered inst.write() for sending commands, inst.read() for receiving responses, and the commonly used inst.query() which combines both.
- **Practical Instrument Interaction:** Applied these methods to set instrument configurations (e.g., voltage range, measurement type) and perform multiple data readings, simulating a typical lab automation workflow.
- **Simulated Instruments:** Utilized PyVISA-py's simulation capabilities to practice without needing physical hardware.

By mastering VISA instrument communication, you are now equipped to automate experiments, collect high-quality data directly from lab equipment, and streamline your test and measurement processes in engineering and scientific research.

ISRO URSC – Python Training | Analog Data | Rajath Kumar

📩 **For queries:** rajath@analogdata.ai| 📱 (+91) 96633 53992 | 🌐 https://analogdata.ai

---
