

Hands-On Guide to LoRa & LoRaWAN with Pycom LoPy4

IntERLab

Abstract

This hands-on session focuses on establishing and configuring Pycom LoPy4 modules for LoRa communication and LoRaWAN integration. Participants will install firmware, write scripts for device-to-device LoRa messaging, and connect LoPy to IntERLab's Private The Things Network (TTN). The goal is to ensure that attendees can successfully implement basic LoRa and LoRaWAN functionality.

Contents

1	Introduction	2
1.1	Objective	2
1.2	Prerequisites	2
1.3	Hands-On Duration	2
1.4	Required Materials	2
1.5	Github Link	2
2	Configuration	3
2.1	Configuring Tools on a Personal Device	3
2.2	Updating the Firmware (Optional)	3
2.3	Connecting to the LoPy Device	4
2.3.1	Executing Hello World on LoPy with MicroPython	5
2.3.2	Failure to connect	5
3	Hands-on LoRa Communication Between LoPy Devices	7
3.1	Creating a New Project	7
3.1.1	Downloading the Pysense Shield Library	7
3.2	Transmitter Code	8
3.3	Receiver Code	9
3.4	Raw LoRa Parameters	10
4	Connecting and Sending Data to LoRaWAN Network	12
4.1	The Private Things Network Stack	12
4.1.1	Creating an Application on The Things Network	12
4.1.2	Registering a Device	13
4.2	Connecting an End Device and Sending Data	13
4.3	Receiving Data from the TTN MQTT Server on Your Laptop	15
4.3.1	Generating an API Key for MQTT	15
4.3.2	Receiving Data as an MQTT Client	16

1 Introduction

1.1 Objective

At the end of this 3-hour workshop, participants will:

- Set up a working environment for running the hands-on session on their device.
- Set up and configure a LoPy module for LoRa communication.
- Send and receive messages between two LoPy devices using LoRa.
- Connect a LoPy device to IntERLab's Private LoRaWAN network using The Things Network (TTN).
- Send data from the LoPy device to the attendee's laptop.

1.2 Prerequisites

- Basic knowledge of Python.
- Familiarity with microcontrollers and wireless communication.
- Personal Laptop to participate in hands-on.

1.3 Hands-On Duration

Task	Duration
Configuring tools on personal device	30 min
Connecting to LoPy Device and updating Firmware	30 min
Hands on LoRa communication between LoPy devices	1 hr
Connecting and Sending Data to LoRaWAN network using OTAA	45 min
Q&A	15 min

The expected duration of the hands-on session is 3 hours.

1.4 Required Materials

- LoPy4 board
- Micro-USB cable
- Computer with Python installed
- Wi-Fi access (for connecting to TTN)

1.5 Github Link

The source code in this hands-on is available at <https://github.com/rmukhia/lora-lab-session>.

2 Configuration

2.1 Configuring Tools on a Personal Device

This section outlines the steps required to install the PyMakr plugin from Pycom in the Visual Studio Code (VSCode) Integrated Development Environment (IDE).

The PyMakr plugin enables users to connect to and program Pycom devices.

Step 1. Download and install Visual Studio Code from the official website: <https://code.visualstudio.com/Download>.

Step 2. Download and install the latest Long-Term Support (LTS) version of Node.js from the official Node.js website: <https://nodejs.org/en>.

Step 3. Open the Extensions View in VSCode by selecting the Extensions icon in the left sidebar or using the shortcut *Ctrl + Shift + X*.

Step 4. In the Extensions search bar, enter *PyMakr*.

Step 5. Click on the *Install* button next to the PyMakr extension to complete the installation.

Step 6. Close VSCode.

For detailed and official instructions, please refer to the Pycom documentation available at <https://docs.pycom.io/gettingstarted/software/vscode/>.

2.2 Updating the Firmware (Optional)

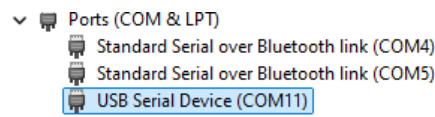
This section describes the process of updating the firmware on a LoPy 4 device. This step can be skipped if the device is already running the latest firmware.

Firmware is low-level software programmed into a hardware device to control its functionality. It acts as an intermediary between the hardware and user applications. Updating the firmware ensures compatibility with new features and security updates.

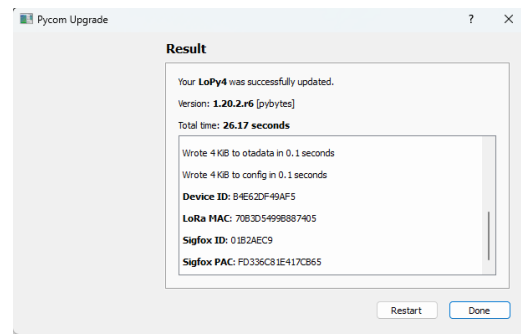
Step 1. Download and install the firmware update tool from <https://docs.pycom.io/updatefirmware/device/>.

Step 2. The tool usually detects the port automatically. If unsure, for Windows open *Device Manager* and check for *USB Serial Device* under the *Ports (COM & LPT)* section, as shown in Figure 1a. For Unix based system run:

```
# ls /dev/tty*
```



(a) Device Manager listing the COM port of the device.



(b) Firmware upgrade completed.

Figure 1: Upgrading firmware.

Step 3. Set the *Type* to *pybytes* for this hands-on session.

Step 4. Check the box *Show Advanced Settings*. This option is useful if the code gets stuck before reaching the REPL.

Step 5. Skip the Pybytes registration for this hands-on session.

Step 6. To clear the filesystem, select *Erase during update* in the Advanced Settings screen. This will delete any existing code on the device.

Step 7. Wait for the firmware upgrade to complete.

Step 8. Take note of the *Device ID* and *LoRa MAC* from the result screen, as shown in Figure 1b, for future reference.

Step 9. Close the tool by pressing *Done*.

2.3 Connecting to the LoPy Device

The following steps describe how to connect the LoPy device to VSCode and upload firmware:

Step 1. Open VSCode and navigate to PyMakr View (Figure 2).

Step 2. Hover over the device (*ttyACMx* in Unix, *COMxx* in Windows) and click *Connect Device*.

Step 3. If the terminal is not visible, click *Create Terminal*.

Step 4. The MicroPython REPL (Read-Eval-Print Loop) will be displayed.

MicroPython is a lightweight implementation of Python designed for microcontrollers. LoPy uses MicroPython to execute scripts, allowing users to write Python code for controlling LoRa communication without needing low-level programming. This simplifies the development and testing of IoT applications.

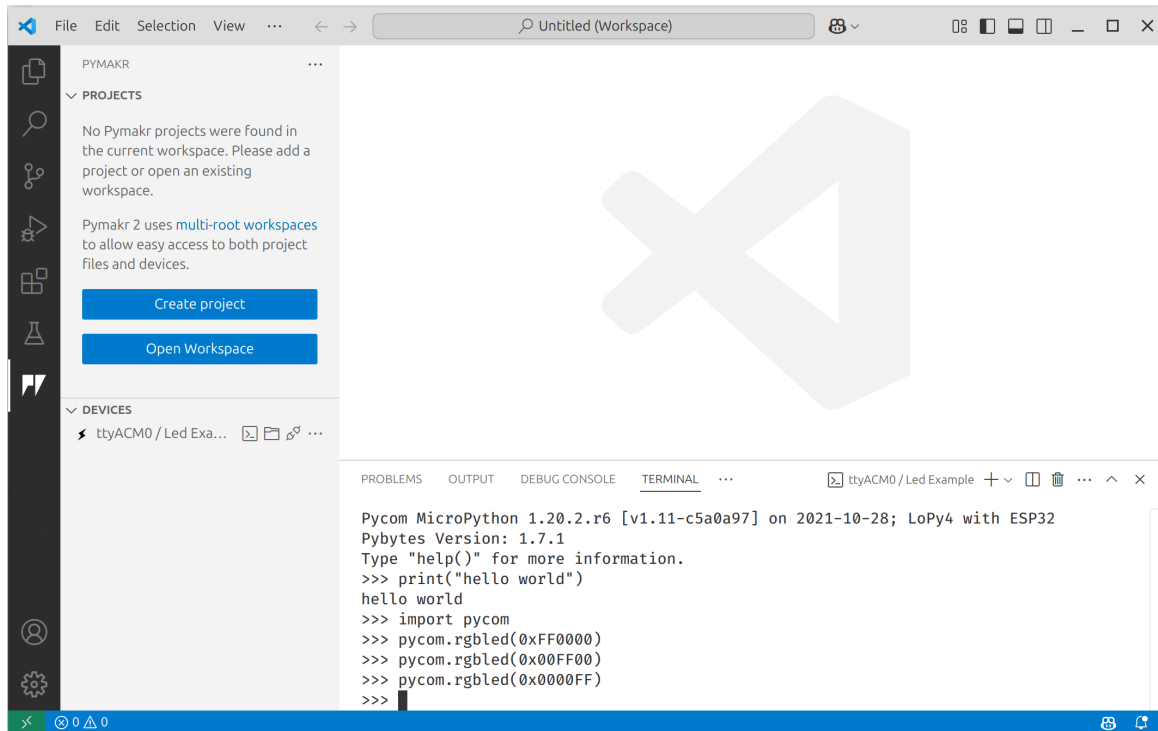


Figure 2: PyMakr View in VSCode.

2.3.1 Executing Hello World on LoPy with MicroPython

Python code can be run in the MicroPython REPL. Try the following commands:

```
>>> print("hello world")
hello world
>>> import pycom
>>> pycom.heartbeat(False)
>>> pycom.rgbled(0xFF0000)
>>> pycom.rgbled(0x00FF00)
>>> pycom.rgbled(0x0000FF)
>>> import ubinascii
>>> import machine
>>> import network
>>> device_id = machine.unique_id()
>>> deveui = network.LoRa().mac()
>>> print("Device id: ", ubinascii.hexlify(device_id).decode())
Device id:  b4e62df49af5
>>> print("DEVEUI: ", ubinascii.hexlify(deveui).decode())
DEVEUI:  70b3d5499b887405
```

2.3.2 Failure to connect

This happens when the LoPy is executing uploaded code in an infinite loop preventing execution to drop to MicroPython REPL.

Pressing *Ctrl + C* on the MicroPython terminal will stop script execution and drop the user to MicroPython REPL. The user should try this if the device is not responding.

When the user has access to the MicroPython terminal, the user can clear the uploaded files using the following snippet.

```
>> import os
>> os.fsformat("/flash")
>> machine.reset()
```

If that does not work, the user can restore the firmware and erase the file system as mentioned in Section 2.2.

3 Hands-on LoRa Communication Between LoPy Devices

In this section, room temperature is read and transmitted using LoRa. Each LoRa packet includes the device ID and the corresponding temperature measured by the device.

3.1 Creating a New Project

Writing on the MicroPython REPL terminal can be convenient, but to perform more complex tasks, we need to create a project and run a Python script on the device.

Step 1. In the Pymakr view, click *Create Project*.

Step 2. Create and select the project folder.

Step 3. Provide a name for the project. The name is stored in the *pymakr.conf* file.

Step 4. Choose *Empty* as the starting template.

Step 5. Select the COM/TTY port of the device.

VSCode will prompt you to trust the project. Accepting this ensures all IDE features are enabled.

The project structure should look like this:

```
Project
|-boot.py  <--- Runs during boot-up
|-main.py  <--- The main Python script of the project
|-pymakr.conf <--- Configuration file
```

3.1.1 Downloading the Pysense Shield Library

Pysense is an expansion shield that provides the following sensors:

- Accelerometer (LIS2HH12)
- Light Sensor (LTR329ALS01)
- Pressure Sensor (MPL3115A2)
- Temperature / Humidity Sensor (SI7006A20)

More details about Pysense can be found at:

<https://docs.pycom.io/datasheets/expansionboards/pysense/>.

Download the *pysense.zip* library from the releases tab of the GitHub repository:

<https://github.com/pycom/pycom-libraries/releases>.

Extract it and copy the *lib* folder to your project. The updated project structure should look like this:

```

Project
|-lib    <--- Folder copied from pysense.zip
|-boot.py <--- Runs during boot-up
|-main.py <--- The main Python script of the project
|-pymakr.conf <--- Configuration file

```

3.2 Transmitter Code

The following code reads and transmits the current temperature along with the device ID. It sends 10 packets, each at an interval of 5 seconds. This code should be placed in *main.py* within the project. Upload the project to the device using the PyCOM view in VSCode.

```

import time
from pycoproc_1 import Pycoproc
import machine
from network import LoRa
import socket
import ubinascii
from SI7006A20 import SI7006A20

py = Pycoproc(Pycoproc.PYSENSE)
si = SI7006A20(py)

def get_temp():
    """
    Retrieves the current temperature.

    Returns:
        float: The current temperature in Celsius.
    """
    temperature = si.temperature()
    print("Temperature: " + str(temperature) + " C")
    return temperature

# Initialize LoRa in LORA mode for the AS923 region
lora = LoRa(mode=LoRa.LORA, region=LoRa.AS923)

# Create a raw LoRa socket
s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)

# Set the socket to non-blocking mode
s.setblocking(False)

# Initialize a counter
i = 0

# Get the unique device ID
device_id = ubinascii.hexlify(machine.unique_id()).decode()

```



```

# Loop to send temperature data 10 times
while i < 10:
    # Get the current temperature
    temp = get_temp()

    # Create a packet with the device ID and temperature
    packet = "{}:{}".format(device_id, temp)

    # Send the packet via LoRa
    s.send(packet)

    # Print the sent packet
    print("Sending {}".format(packet))

    # Increment the counter
    i += 1

    # Wait for 5 seconds before sending the next packet
    time.sleep(5)

```

3.3 Receiver Code

The following code receives the LoRa packet and prints the contents along with the RSSI and SNR values. This should be placed in *main.py* in a separate project from the Transmitter Code.

```

from network import LoRa
import socket
import time

# Initialize LoRa in LORA mode for the AS923 region
lora = LoRa(mode=LoRa.LORA, region=LoRa.AS923)

# Create a raw LoRa socket
s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)

# Set the socket to non-blocking mode
s.setblocking(False)

# Infinite loop to continuously check for incoming messages
while True:
    # Receive data from the socket (up to 64 bytes)
    rx = s.recv(64)

    # If data is received, process it
    if len(rx) > 0:
        # Print the received message and LoRa statistics
        print("Received message:", rx.decode(),

```

```

    "| RSSI:", lora.stats().rssi, "| SNR:", lora.stats().snr)

# Sleep for 1 second before checking again
time.sleep(1)

```

3.4 Raw LoRa Parameters

LoRa frequency varies by region and is regulated to ensure interference-free operation. Common frequency bands include EU868 (Europe), US915 (United States), and AS923 (Asia-Pacific). It is essential to set the correct frequency region in the LoPy code to comply with local regulations.

For raw LoRa mode, the following parameters can be adjusted in LoPy:

Region Settings Available region parameters:

- LoRa.AS923
- LoRa.AU915
- LoRa.US915
- LoRa.CN470
- LoRa.IN865

Frequency Configuration The frequency can be configured, but it must comply with region-specific limits.

Coding Rate The coding rate can be set to:

- LoRa.CODING_4_5
- LoRa.CODING_4_6
- LoRa.CODING_4_7
- LoRa.CODING_4_8

Bandwidth Configuration Available bandwidth settings:

- LoRa.BW_125KHZ
- LoRa.BW_250KHZ
- LoRa.BW_500KHZ

Preamble Length The default is set to 8 symbols.

Spreading Factor (SF) Can be set between 7 and 12.

Power Modes Power mode options:

- LoRa.ALWAYS_ON - The radio is always listening.
- LoRa.TX_ONLY - The radio sleeps after transmission.
- LoRa.SLEEP - The radio remains off until reactivated.

Try to change constructor parameters of the LoRa class and experiment.

```
LoRa(mode, region=LoRa.EU868, frequency, bandwidth=LoRa.BW_125KHZ,  
      sf=7, preamble=8, coding_rate=LoRa.CODING_4_5, power_mode=LoRa.ALWAYS_ON)
```

4 Connecting and Sending Data to LoRaWAN Network

4.1 The Private Things Network Stack

IntERLab maintains a private instance of The Things Network Stack (TTN) for LoRaWAN communication. Users must create an application on the platform and then register a device in that application. Multiple devices can be registered under the same application. In this hands-on exercise, each user will create an application and register their device under it.

Comprehensive details about LoRaWAN are available at <https://www.thethingsnetwork.org/docs/lorawan/>.

Register end device

From The LoRaWAN Device Repository [Manually](#)

Frequency plan ⓘ *

Asia 923-925 MHz

LoRaWAN version ⓘ *

LoRaWAN Specification 1.0.2

Regional Parameters version ⓘ *

RP001 Regional Parameters 1.0.2 revision B

[Show advanced activation, LoRaWAN class and cluster settings](#) ▾

DevEUI ⓘ *

70 B3 D5 49 9A 2C 58 6A

AppEUI ⓘ *

00 00 00 00 00 00 00 00 [Fill with zeros](#)

AppKey ⓘ *

80 48 D9 E1 C1 D2 14 4E 49 07 1F 7C 81 52 6D EE [Generate](#)

End device ID ⓘ *

lopy-120049

This value is automatically prefilled using the DevEUI

After registration

☒ View registered end device

☐ Register another end device of this type

[Register end device](#)

Figure 3: Registering an end device

4.1.1 Creating an Application on The Things Network

Step 1. Log in to *tn.hazemon.in.th* using the user credentials provided.

Step 2. Navigate to the *Applications* tab.

Step 3. Click on the *Add Application* button.

Step 4. Provide an *Application ID* (required), an *Application Name* (optional), and a *Description* (optional), then create the application. The application ID should be a unique and memorable name, such as your student ID.

4.1.2 Registering a Device

The next step is to add devices to the application.

Step 1. Retrieve the *Device EUI* of your device. Refer to Section 2.3.

Step 2. Click on *Add end device* in the *Overview* or *End Device* section.

Step 3. Add the device *manually*. Refer to Figure 3 to fill in the required parameters. The *DevEUI* should be the ID obtained from Step 1. The *AppEUI* can be filled with zeros and the *AppKey* can be generated. The *End device ID* should be a unique device name.

By completing the above steps, the device will be registered as a Class A device. Additional details about device classes are available at <https://www.thethingsnetwork.org/docs/lorawan/classes/>.

4.2 Connecting an End Device and Sending Data

The following *main.py* script joins the end device to the TTN network using OTAA and reads and transmits the temperature 100 times.

Further information on OTAA and ABP is available at <https://www.thethingsindustries.com/docs/hardware/devices/concepts/abp-vs-otaa/>.

The *app_eui* and *app_key* must be set according to the values generated in Section 4.1.2, Step 3.

During execution, activity can be observed in the *Live data* section of the TTN application. This script should be uploaded to the end device.

```
import time
from pycoproc_1 import Pycoproc
import machine
from network import LoRa
import socket
import ubinascii
from SI7006A20 import SI7006A20

# Initialize the Pycoproc object for the Pysense board
py = Pycoproc(Pycoproc.PYSENSE)

# Initialize the SI7006A20 sensor with the Pycoproc object
si = SI7006A20(py)

def get_temp():
    """
    Retrieves the current temperature.

    Returns:
        float: The current temperature in Celsius.
    """
```

```
# Get the temperature from the SI7006A20 sensor
temperature = si.temperature()

# Print the temperature
print("Temperature: " + str(temperature) + " C")

# Return the temperature
return temperature

# Initialize LoRa in LORAWAN mode for the Asia region
lora = LoRa(mode=LoRa.LORAWAN, region=LoRa.AS923)

# Create an OTAA authentication parameters
app_eui = ubinascii.unhexlify("0000000000000000")
app_key = ubinascii.unhexlify("0939E230785BEAC72F4994467CD490D1")

# Join a network using OTAA (Over the Air Activation)
lora.join(activation=LoRa.OTAA, auth=(app_eui, app_key), timeout=0)

# Wait until the module has joined the network
while not lora.has_joined():
    # Sleep for 2.5 seconds before checking again
    time.sleep(2.5)
    # Print a message indicating that the module has not yet joined
    print("Not yet joined...")

# Create a LoRa socket
s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)

# Set the LoRaWAN data rate
s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)

# Make the socket blocking
# (waits for the data to be sent and for the 2 receive windows to expire)
s.setblocking(True)

# Initialize a counter
i = 0

# Loop to send temperature data 100 times
while i < 100:
    # Get the current temperature
    temp = get_temp()

    # Create a packet with the device ID and temperature
    packet = "Temperature: {}".format(temp)
```

```
# Send the packet via LoRa
s.send(packet)

# Print the sent packet
print("Sending {}".format(packet))

# Increment the counter
i += 1

# Wait for 5 seconds before sending the next packet
time.sleep(5)
```

4.3 Receiving Data from the TTN MQTT Server on Your Laptop

The next step is to receive data from the end device on your laptop. There are multiple ways to receive data, such as MQTT, Webhooks, etc. However, for this hands-on exercise, we will focus on receiving data via the TTN MQTT server. A detailed document regarding integration can be accessed at <https://www.thethingsindustries.com/docs/integrations/>.

4.3.1 Generating an API Key for MQTT


Step 1. Navigate to *Integrations* → *MQTT*.

Step 2. Click on *Generate new API key*. Copy and save the API key as it will be needed to connect to the MQTT server.

MQTT

MQTT is a publish/subscribe messaging protocol designed for IoT. Every application on TTS automatically exposes an MQTT endpoint. In order to connect to the MQTT server you need to create a new API key, which will function as connection password. You can also use an existing API key, as long as it has the necessary rights granted.

Further resources

 [MQTT server](#) | [Official MQTT website](#)

Connection information

MQTT server host

Public address

Public TLS address

Connection credentials

Username

Password

Figure 4: TTN MQTT Credentials

4.3.2 Receiving Data as an MQTT Client

Create a Python project, open it in VSCode and install *paho-mqtt* from <https://pypi.org/project/paho-mqtt/>.

Open the VSCode terminal for your project. It is recommended to create a Python virtual environment and install *paho-mqtt* within it to avoid installing it as a global package.

Activating the Virtual Environment

```
$ python -m venv venv
```

Choose one of the following:

```
$ source venv/bin/activate # for Linux, macOS
```

```
$ venv/Scripts\activate.bat # for Windows (cmd.exe)
```

```
$ venv/Scripts\Activate.ps1 # for Windows (PowerShell)
```

```
$ pip install paho-mqtt
```

MQTT Client The following script connects to the TTN MQTT server and handles incoming messages. Set *USERNAME* and *PASSWORD* to the values obtained in Figure 4. Also, set *APP_NAME* and *DEVICE_ID* to the application ID and device ID specified in Section 4.1.1 and Section 4.1.2, respectively.

```
import paho.mqtt.client as mqtt
import json
import base64

def on_message(client, userdata, msg):
    # Decode the JSON payload from the message
    data = json.loads(msg.payload.decode())
    # Extract the base64 encoded payload
    payload_base64 = data["uplink_message"]["frm_payload"]

    # Decode the base64 payload to a UTF-8 string
    payload = base64.b64decode(payload_base64).decode()

    # Print the received message
    print(f"Received message: {payload}")

# MQTT client setup
client = mqtt.Client(callback_api_version=mqtt.CallbackAPIVersion.VERSION2)
# Set the on_message callback function
client.on_message = on_message

# MQTT broker username
USERNAME = "testapp"
# MQTT broker password
```



```
PASSWORD = "NNSXS.2MU7ZTKELP3KSVXBWU"

# Set the username and password for the MQTT client
client.username_pw_set(USERNAME, PASSWORD)

# MQTT broker address
BROKER = "ttn.hazemon.in.th"
# MQTT broker port
PORT = 1883

# Connect to the MQTT broker
client.connect(BROKER, PORT)

# Application name
APP_NAME = "testapp"
# Device ID
DEVICE_ID = "lopy-120049"

# Subscribe to the topic for the device uplink messages
client.subscribe(f"v3/{APP_NAME}/devices/{DEVICE_ID}/up")

# Start the MQTT client loop to process messages
client.loop_forever()

Run this script using the following command on the VSCode terminal:

$ python main.py
```