Lab: MicroPython: Machine Learning and MQTT

1. Introduction

The objective of this practice is to further explore the functionality that the Arduino nano rp2040 can offer. For this we will explore how to use pre-trained machine learning models.

1.1 MicroPython

Since its launch in 1991, the Python programming language – named after the famous comedy troupe Monty Python, rather than the snake – has grown to become one of the most popular in the world. Its popularity, though, doesn't mean there aren't improvements that could be made – particularly if you're working with a microcontroller. The Python programming language was developed for computer systems like desktops, laptops, and servers. Microcontroller boards like Raspberry Pi Pico are smaller, simpler, and with considerably less memory – meaning they can't run the same Python language as their bigger counterparts. That's where MicroPython comes in. Originally developed by Damien George and first released in 2014, MicroPython is a Python-compatible programming language developed specifically for microcontrollers. It includes many of the features of mainstream Python, while adding a range of new ones designed to take advantage of the facilities available on Raspberry Pi Pico and other microcontroller boards. If you've programmed with Python before, you'll find MicroPython immediately familiar. If not, don't worry: it's a friendly language to learn!

A MicroPython program, just as with a standard Python program, normally runs top-to-bottom: it goes through each line in turn, running it through the interpreter before moving on to the next, just as if you were typing them line-by-line into the Shell. A program that just runs through a list of instructions line-by-line wouldn't be very clever, though – so MicroPython, just like Python, has its own way of controlling the sequence in which its programs run: *indentation*.

If this is the first time you use Python, please check: https://www.w3schools.com/python/default.asp

1.2 OpenMV

Arduino has made some unusual choices with MicroPython, in particular, the use of OpenMV for programming the Arduino boards. This is unusual as OpenMV is normally used with cameras and machine vision, and not for generic microcontroller boards. It does however provide a way to install MicroPython and upload new programs.

Exercises

Exercise 0 – SET UP

To work with micropython we need to use a non-Arduino IDE. **Please follow the instructions in this tutorial and configure the Arduino Nano board to be detected by OpenMV**.

The second step in the configuration is to update the Nina firmware in your arduino board. **We will need to use the Arduino IDE for this step. https://docs.arduino.cc/tutorials/nano-rp2040-connect/rp2040-upgrading-nina-firmware.** You will see that there are three options offered for the update: (Option 1) over the Arduino Cloud, (Option 2) using the Arduino IDE (https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-fw-cert-uploader#firmware-updater) or (Option 3) with a firmware loader. We recommend you to use Option 2, but you can also try Option 1. If you choose Option 2, you can go directly to the second link. Follow the instructions to update the firmware (not to upload certificates). Please be aware that to work again with the board in the Arduino IDE after being connected to the OpenMV, you will first have to:

- Go to OpenMV, stop the execution of the program (red button) and close the Open MV
- Open Arduino IDE and double-tap again on the reset-button on the board to enter bootloader

Exercise 1 – MACHINE LEARNING MODELS

Next we are going to reuse a machine learning model to start developing a music detector. **Please follow this tutorial and modify the code so that instead of detecting vibration, the Arduino detects: no movement, slow dance and fast dance. The strings to be printed should be "dance|NO_DANCING", "dance|SLOW_DANCING" and "dance|FAST_DANCING".** Before starting with the tutorial, download the ML libraries and store them into the Arduino flash memory (the disk that opens when double-tapping, maybe called "NO NAME"):

- lsm6dsox.py
- lsm6dsox_vibration_monitoring.ucf

Warnings: if you copy-paste the example, you may get an "ENOENT" error. Check that the editor has your code and not the library open. Go to the window of your code and remove the last trailing end-of-line. Run the example again.

Submit: Exercise1_ML.py

Exercise 2 – MQTT COMUNICATION – PUBLIC BROKER

What is MQTT?

MQTT is a publish-subscribe-based messaging protocol used in the internet of Things. It works on top of the TCP/IP protocol, and is designed for connections with remote locations where a "small code footprint" is required or the network bandwidth is limited. The MQTT protocol provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for Internet of Things messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers.

The next step in the development of our dance detector is to communicate the messages via MQTT.

We suggest that you read this tutorial: https://www.tomshardware.com/how-to/send-and-receive-data-raspberry-pi-pico-w-mqtt

For this purpose we use the HiveMQ Public MQTT Broker to access the messages as a client. For example you can use the MQTT Browser Client (http://www.hivemq.com/demos/websocket-client/) to receive messages. In order to send and receive messages via MQTT, you can use the WiFi library of Exercise 1 and any mqtt library provided for micropython. For example micropython-umqtt.simple or micropython-umqtt.robust.

You have to combine both Exercise 1 (with the ML model) and the example of the MQTT tutorial. Instead of sending the sensor value, you send the detected dance movement. You can freely select the name of the topic for now.

Submit: Exercise2_ML_MQTT.py