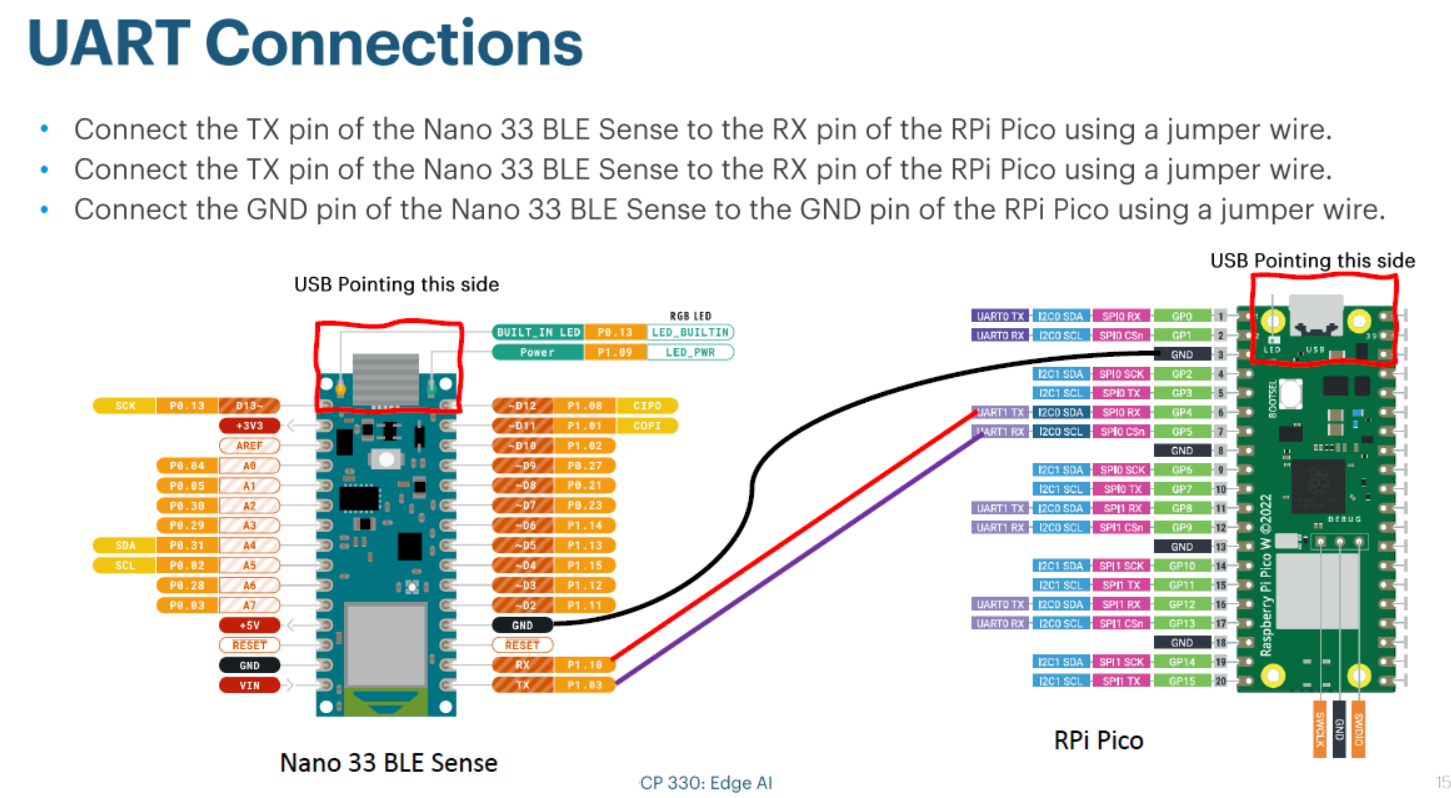
S-1) insert open mv firmware in nano ble and micropython firmware in pico.

S-2) do the wire connection



S-3) Run following code on thonny ( connected with nano ble)

import time, math

from machine import UART

from lsm9ds1 import LSM9DS1

from machine import Pin, I2C

uart = UART(0, baudrate=9600)

def score(input):

if input[4] <= 0.3368529975414276:

if input[2] <= -0.29931640625:

var0 = [0.0, 0.0, 1.0, 0.0, 0.0]

else:

if input[12] <= 0.1018066480755806:

if input[14] <= -0.04895019344985485:

if input[3] <= 0.360351599752903:

if input[3] <= -0.5415649395436049:

var0 = [0.0, 0.0, 1.0, 0.0, 0.0]

else:

var0 = [0.0, 1.0, 0.0, 0.0, 0.0]

else:

if input[11] <= 0.7155395448207855:

var0 = [0.0, 0.0, 0.0, 1.0, 0.0]

else:

var0 = [1.0, 0.0, 0.0, 0.0, 0.0]

else:

var0 = [0.0, 1.0, 0.0, 0.0, 0.0]

else:

if input[2] <= 0.4770202785730362:

var0 = [0.0, 0.0, 0.0, 0.0, 1.0]

else:

var0 = [0.0, 0.0, 0.0, 1.0, 0.0]

else:

if input[8] <= 0.458892822265625:

if input[14] <= 0.02947998046875:

var0 = [1.0, 0.0, 0.0, 0.0, 0.0]

else:

var0 = [0.0, 0.0, 1.0, 0.0, 0.0]

else:

if input[9] <= 0.9161987006664276:

var0 = [0.0, 0.0, 0.0, 1.0, 0.0]

else:

var0 = [0.0, 1.0, 0.0, 0.0, 0.0]

return var0

import math

class SensorDataProcessor:

method\_names = ['mean', 'std\_dev', 'min', 'max', 'median', 'var', 'range', 'iqr', 'rms', 'skew', 'kurtosis']

max\_length = 10

def \_\_init\_\_(self, prefix):

self.buffer = []

self.prefix\_name = prefix

def add\_data(self, new\_data):

"""Adds new data to the buffer, maintaining max length."""

if len(self.buffer) == self.max\_length:

self.buffer.pop(0)

self.buffer.append(new\_data)

def mean(self, buffer):

return sum(buffer) / len(buffer)

def std\_dev(self, buffer):

mean\_val = self.mean(buffer)

return math.sqrt(sum((x - mean\_val) \*\* 2 for x in buffer) / (len(buffer) - 1))

def min(self, buffer):

return float(min(buffer))

def max(self, buffer):

return float(max(buffer))

def median(self, buffer):

"""Computes the median manually."""

sorted\_buffer = sorted(buffer)

n = len(sorted\_buffer)

mid = n // 2

if n % 2 == 0:

return (sorted\_buffer[mid - 1] + sorted\_buffer[mid]) / 2

else:

return sorted\_buffer[mid]

def var(self, buffer):

"""Computes variance manually."""

mean\_val = self.mean(buffer)

return sum((x - mean\_val) \*\* 2 for x in buffer) / (len(buffer) - 1)

def range(self, buffer):

"""Computes range: max - min."""

return float(max(buffer) - min(buffer))

def iqr(self, buffer):

"""Computes Interquartile Range (IQR) manually."""

sorted\_buffer = sorted(buffer)

n = len(sorted\_buffer)

mid = n // 2

if n % 2 == 0:

q1 = self.median(sorted\_buffer[:mid])

q3 = self.median(sorted\_buffer[mid:])

else:

q1 = self.median(sorted\_buffer[:mid])

q3 = self.median(sorted\_buffer[mid + 1:])

return q3 - q1

def rms(self, buffer):

"""Computes Root Mean Square (RMS)."""

return math.sqrt(sum(x \*\* 2 for x in buffer) / len(buffer))

def skew(self, buffer):

"""Computes skewness manually."""

n = len(buffer)

mean\_val = self.mean(buffer)

std\_val = self.std\_dev(buffer)

if std\_val == 0:

return 0 # Avoid division by zero

return sum((x - mean\_val) \*\* 3 for x in buffer) / (n \* (std\_val \*\* 3))

def kurtosis(self, buffer):

"""Computes excess kurtosis manually (subtracting 3 for normal distribution)."""

n = len(buffer)

mean\_val = self.mean(buffer)

std\_val = self.std\_dev(buffer)

if std\_val == 0:

return 0 # Avoid division by zero

return sum((x - mean\_val) \*\* 4 for x in buffer) / (n \* (std\_val \*\* 4)) - 3

def print\_feature\_names(self):

"""Prints the names of extracted features."""

feature\_names = [f"{self.prefix\_name}\_{i}" for i in range(self.max\_length)]

feature\_names += [f"{self.prefix\_name}\_{method}" for method in self.method\_names]

print(feature\_names)

def calculate\_all(self, new\_data):

"""Computes all statistical features for the buffer."""

self.add\_data(new\_data)

if len(self.buffer) < self.max\_length:

return

results = self.buffer.copy() # Store raw buffer values

for method\_name in self.method\_names:

method = getattr(self, method\_name)

results.append(method(self.buffer))

return results

# Initialize the sensor (IMU) and I2C bus.

i2c = I2C(1, scl=Pin(15), sda=Pin(14))

imu = LSM9DS1(i2c)

# Create SensorDataProcessor objects for accelerometer, gyroscope, and magnetometer axes.

# Accelerometer processors

acc\_x\_obj = SensorDataProcessor(prefix="ax\_")

acc\_y\_obj = SensorDataProcessor(prefix="ay\_")

acc\_z\_obj = SensorDataProcessor(prefix="az\_")

# Gyroscope processors

gyro\_x\_obj = SensorDataProcessor(prefix="gx\_")

gyro\_y\_obj = SensorDataProcessor(prefix="gy\_")

gyro\_z\_obj = SensorDataProcessor(prefix="gz\_")

# Magnetometer processors

mag\_x\_obj = SensorDataProcessor(prefix="mx\_")

mag\_y\_obj = SensorDataProcessor(prefix="my\_")

mag\_z\_obj = SensorDataProcessor(prefix="mz\_")

activity\_labels = {

0: "Idle",

1: "Jump",

2: "type",

3: "walk",

4: "write"

}

selected\_features = [

("mx\_max", mag\_x\_obj, "max"),

("az\_mean", acc\_z\_obj, "mean"),

("az\_median", acc\_z\_obj, "median"),

("az\_min", acc\_z\_obj, "min"),

("ax\_min", acc\_x\_obj, "min"),

("my\_median", mag\_y\_obj, "median"),

("mx\_mean", mag\_x\_obj, "mean"),

("my\_rms", mag\_y\_obj, "rms"), # RMS = sqrt(mean\_square)

("ax\_mean", acc\_x\_obj, "mean"),

("az\_max", acc\_z\_obj, "max"),

("my\_max", mag\_y\_obj, "max"),

("my\_mean", mag\_y\_obj, "mean"),

("mz\_median", mag\_z\_obj, "median"),

("my\_min", mag\_y\_obj, "min"),

("mx\_min", mag\_x\_obj, "min"),

("mx\_median", mag\_x\_obj, "median"),

("mz\_max", mag\_z\_obj, "max"),

("mz\_mean", mag\_z\_obj, "mean"),

("mz\_min", mag\_z\_obj, "min"),

("az\_var", acc\_z\_obj, "var"),

]

while True:

# Get raw sensor data.

ax, ay, az = imu.accel()

gx, gy, gz = imu.gyro()

mx, my, mz = imu.magnet()

# Print raw sensor data for debugging.

#print("Accel:", ax, ay, az)

#print("Gyro:", gx, gy, gz)

#print("Mag:", mx, my, mz)

# Process each reading.

ax\_result = acc\_x\_obj.calculate\_all(ax)

ay\_result = acc\_y\_obj.calculate\_all(ay)

az\_result = acc\_z\_obj.calculate\_all(az)

gx\_result = gyro\_x\_obj.calculate\_all(gx)

gy\_result = gyro\_y\_obj.calculate\_all(gy)

gz\_result = gyro\_z\_obj.calculate\_all(gz)

mx\_result = mag\_x\_obj.calculate\_all(mx)

my\_result = mag\_y\_obj.calculate\_all(my)

mz\_result = mag\_z\_obj.calculate\_all(mz)

if all(obj.buffer is not None and len(obj.buffer) == obj.max\_length for \_, obj, \_ in selected\_features):

# Extract only selected features in the given order

feature\_vector = [getattr(obj, method)(obj.buffer) for \_, obj, method in selected\_features]

# Print the selected feature vector for debugging

#print("Selected Feature Vector:", feature\_vector)

# Predict activity

prediction\_result = score(feature\_vector)

max\_accuracy = max(prediction\_result)

max\_index = prediction\_result.index(max\_accuracy)

activity = activity\_labels[max\_index]

print("Detected Activity:", activity)

uart.write(activity + '\n') # Add newline for easier parsing on receiving end

time.sleep(0.1)

S-4) Run following code on thonny (connected with pico without MQTT):-

from machine import Pin, UART

import time

# Initialize UART1 with GPIO4 (TX) and GPIO5 (RX)

uart = UART(1, baudrate=9600, tx=Pin(4), rx=Pin(5))

# List of expected activity labels (optional filtering)

activities = ['Walking', 'Running', 'Standing', 'Sitting', 'Jumping']

# Initialize buffer

buffer = ""

# Read data from UART

while True:

if uart.any():

char = uart.read(1) # Read one character at a time

if char:

char = char.decode('utf-8')

if char == '\n': # End of message

activity = buffer.strip()

if activity in activities:

print(f"Activity Detected: {activity}")

else:

print(f"Unknown activity received: {activity}")

buffer = "" # Reset buffer

else:

buffer += char

S-5)Next part is connecting with mqqt

from machine import Pin, UART

import time

import network

from umqtt.simple import MQTTClient

import gc

# Initialize UART1 with GPIO4 (TX) and GPIO5 (RX)

uart = UART(1, baudrate=9600, tx=Pin(4), rx=Pin(5))

#connect to wifi

wifi\_ssid = "siddharth\_hotspot"

wifi\_password = "12345789"

wlan = network.WLAN(network.STA\_IF)

wlan.active(True)

wlan.connect(wifi\_ssid, wifi\_password)

while not wlan.isconnected():

print('Waiting for connection...')

time.sleep(1)

print('Connected to WiFi')

# Connect to MQTT Broker

mqtt\_host = "broker.mqtt.cool"

mqtt\_client\_id = "pico\_uart\_mqtt2"

mqtt\_client = MQTTClient(client\_id=mqtt\_client\_id, server=mqtt\_host)

try:

mqtt\_client.connect()

except Exception as e:

print("MQTT connection failed:", e)

time.sleep(2)

# List of expected activity labels (optional filtering)

#activities = ['Walking', 'Running', 'Standing', 'Sitting', 'Jumping']

# Initialize buffer

#buffer = ""

# Read data from UART

while True:

if uart.any():

line = uart.readline()

if line:

line = line.decode('utf-8').strip()

print("Received from Nano:", line)

try:

mqtt\_client.publish("pat/imu", line)

except Exception as e:

print("MQTT publish failed:", e)

# Print available memory

gc.collect()

print("Free memory:", gc.mem\_free())

time.sleep(0.5)