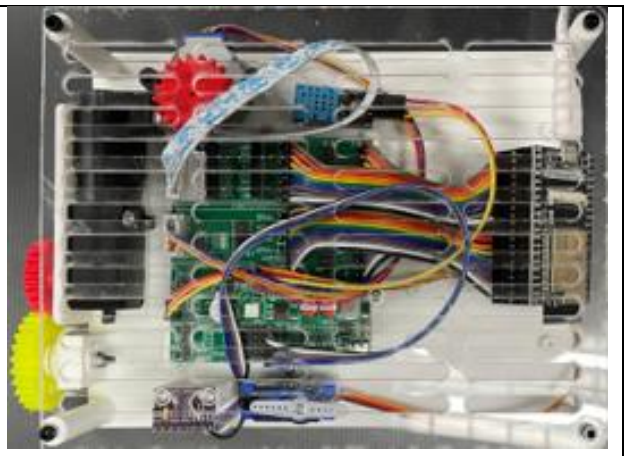
Tutorial 3 ROS Programming - Sensors and Data Visualisation

Objective

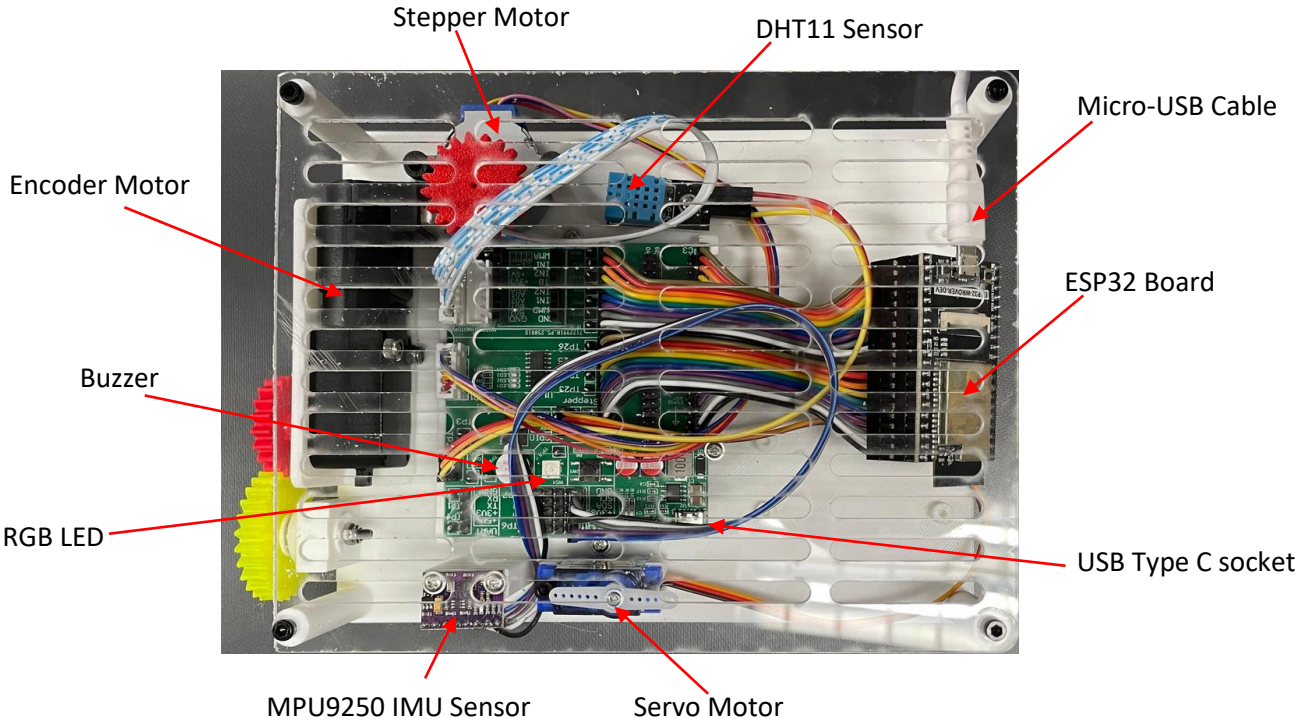
1. Guide students through integrating and programming sensors, specifically the DHT11 temperature and humidity sensor and the MPU9250 IMU, with an ESP32 microcontroller using ROS 2 Jazzy on Ubuntu 24.04.
2. Focus on enabling participants to collect sensor data, publish it via ROS 2 topics, and visualise the data through a Python script, leveraging the Arduino IDE for programming the ESP32.
3. Help students to understand how to interface sensors with ROS 2, process sensor data, and create basic visualisations to monitor environmental and motion data effectively.

1 Introduction to the hardware items

1. A list of hardware items

1	ESP32 sensors and actuators kit	
2	A micro-USB Type-A cable	
3	A USB Type-C cable (If the PD Charger provides a Type-C socket)	
4	A PD Charger	 or 

2. Major components of the ESP32 sensors and actuator kit



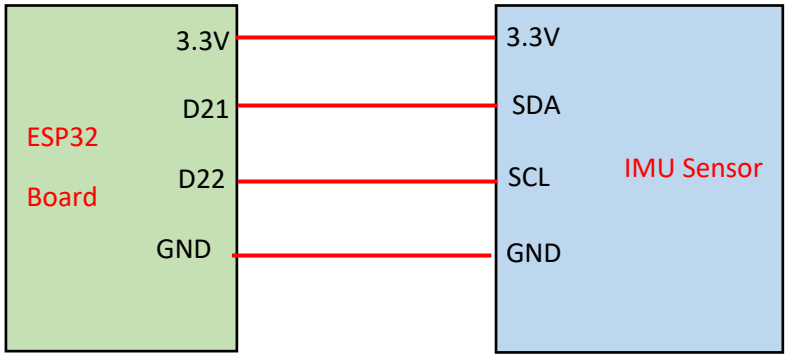
3. Control circuit block diagram for the DHT11 sensor

The data communication of the DHT11 sensor uses 1 digital signal pin. The ESP32 uses digital input/output port number 27 to control the DHT11 sensor.



4. Control circuit block diagram for the MPU9250/MPU6050 IMU sensor

The ESP32 uses the I2C (IIC) bus to control the IMU sensor. The SDA is the data signal of the I2C bus. The SCL is the clock signal of the I2C bus.



2 Install Arduino IDE

1. Download Linux AppImage from Arduino website to ~/Downloads. (https://www.arduino.cc/en/software)
2. Set up the application and run the Arduino IDE:
   Open a new terminal

```
$ mkdir ~/Arduino

$ cd ~/Arduino

$ mv ~/Downloads/arduino-ide_2.3.6_Linux_64bit.AppImage ~/Arduino/

$ chmod +x ~/Arduino/arduino-ide_2.3.6_Linux_64bit.AppImage

$ sudo apt update && sudo apt install libfuse2 -y

$ ./arduino-ide_2.3.6_Linux_64bit.AppImage --appimage-extract

$ cd squashfs-root

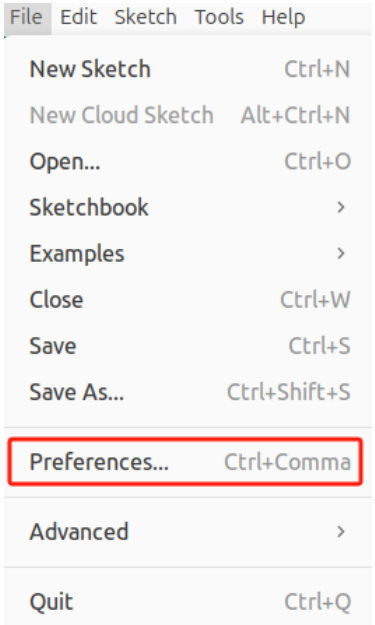
$ sudo chown root:root chrome-sandbox

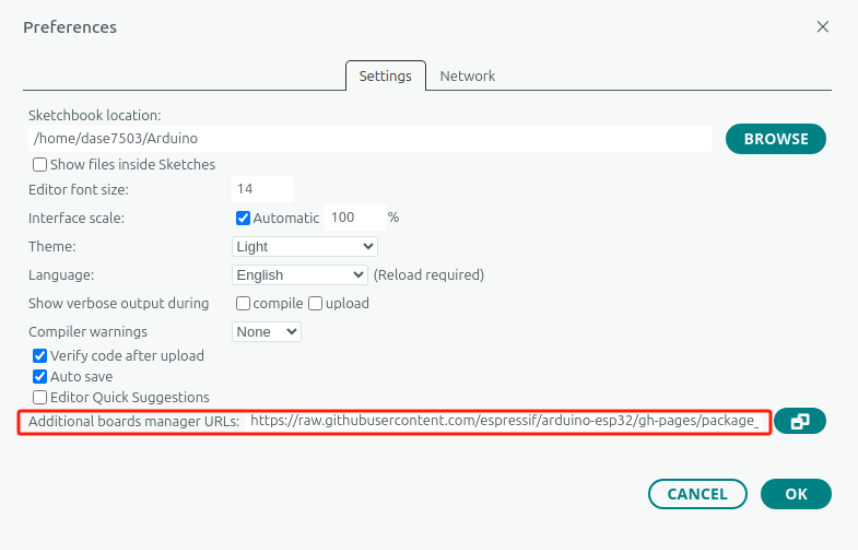
$ sudo chmod 4755 chrome-sandbox

$ ln -s ~/Arduino/squashfs-root/AppRun ~/Arduino/arduino-ide

$ ./AppRun
```

3. Configure the ESP32 support
   In Arduino IDE, add to Preferences > Additional Boards Manager URLs:
   https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json





3 Install micro-ROS Arduino Library

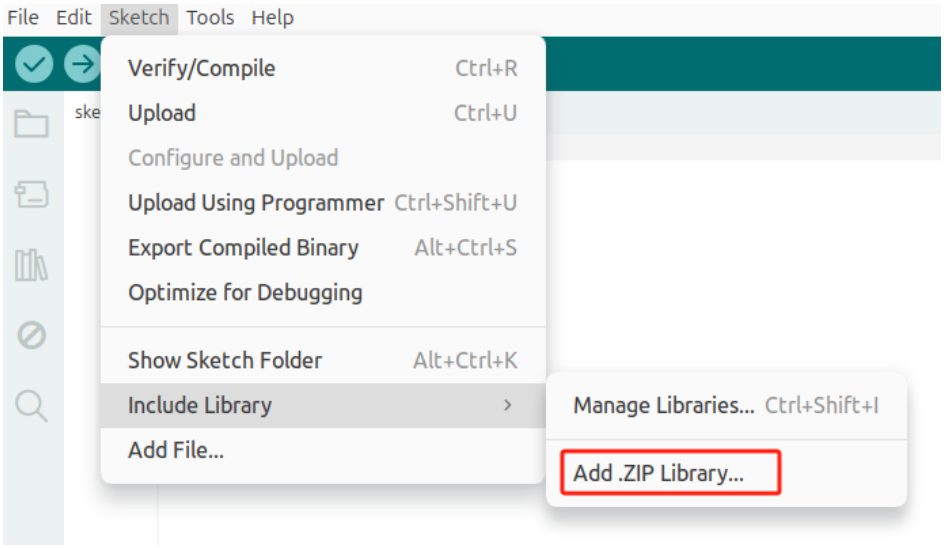
1. Download the micro-ROS Arduino library:
   Open a new terminal

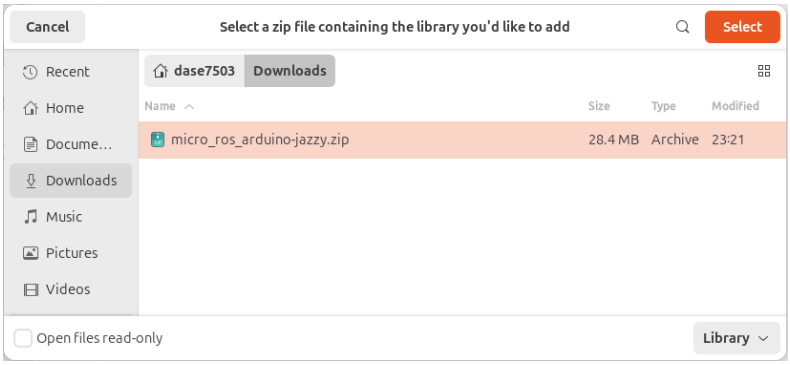
```
$ cd ~/Downloads

$ wget https://codeload.github.com/micro-ROS/micro_ros_arduino/zip/refs/heads/jazzy

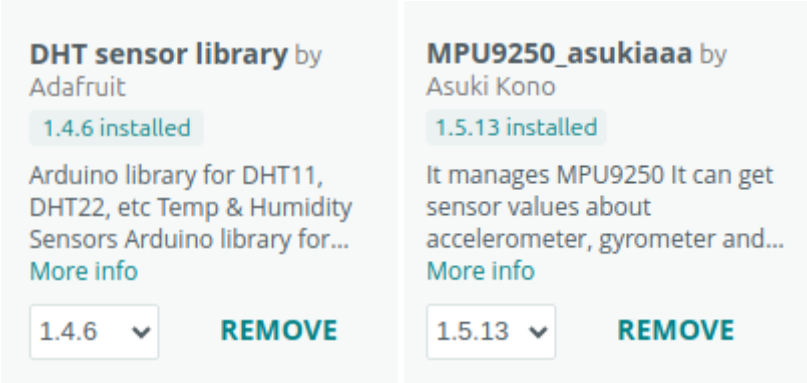
$ mv jazzy micro_ros_arduino-jazzy.zip
```

2. Add the micro-ROS Arduino library:
   Sketch > Include Library > Add .ZIP Library, select ~/Downloads/micro_ros_arduino-jazzy.zip.

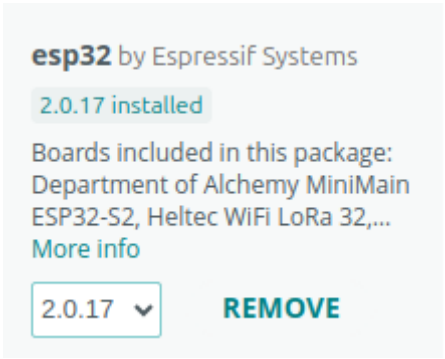




3. In Arduino IDE, Sketch > Include Library > Manage Libraries: Search and install "DHT sensor library" and "MPU9250_asukiaaa".



Install esp32 board library by: Tools > Board Manager, search esp32 (by Espressif Systems) and install it. **Version 2.0.17** is recommended.



4. Install Python dependencies:
   Open a new terminal

```
$ sudo apt install python3-serial python3-pip pipx -y
$ pipx install pyserial esptool
$ pipx ensurepath
$ pipx completions
```

4 Install micro-ROS Workspace for ROS2 Jazzy

1. Install Docker:
   Open a new terminal

```
$ sudo apt install docker.io -y

$ sudo usermod -aG docker $USER
```

2. Create Workspace:

```
$ mkdir -p ~/microros_ws && cd ~/microros_ws

$ git clone -b jazzy https://github.com/micro-ROS/micro_ros_setup.git
src/micro_ros_setup

$ sudo apt update && rosdep update

$ sudo rosdep init && rosdep update

$ rosdep install --from-paths src --ignore-src -y

$ source /opt/ros/jazzy/setup.bash

$ colcon build

$ source install/local_setup.bash
```

5 Test the DHT11 sensor on ESP32

What is DHT11?

The DHT11 is a low-cost, digital temperature and humidity sensor widely used in electronics projects for environmental monitoring. It combines a thermistor for temperature measurement and a capacitive humidity sensor, providing digital output via a single-wire interface, making it easy to integrate with microcontrollers like Arduino or ESP32.

1. In Arduino IDE, create a new sketch by File>New Sketch
2. Include the necessary libraries and define the pin number of DHT11 by adding these codes into the sketch:

```
#include <DHT.h>

#include <micro_ros_arduino.h>

#include <rcl/rcl.h>

#include <rclc/rclc.h>

#include <rclc/executor.h>

#include <sensor_msgs/msg/temperature.h>

#include <sensor_msgs/msg/relative_humidity.h>

#define DHT_PIN 27

#define DHT_TYPE DHT11
```

   a) DHT_PIN: The GPIO pin connected to the DHT11 sensor.
   b) DHT_TYPE: Specifies the sensor model (DHT11).

3. Implement macros for error checking in micro-ROS by adding these codes:

```
#define RCCHECK(fn, context) { rcl_ret_t temp_rc = fn; if(temp_rc !=
RCL_RET_OK){return;}}

#define RCSOFTCHECK(fn, context) { rcl_ret_t temp_rc = fn; if(temp_rc !=
RCL_RET_OK){}}
```

4. Initialize the DHT11 sensor and test basic readings by adding this code:

```
DHT dht(DHT_PIN, DHT_TYPE);
```

5. Define micro-ROS variables and implement the timer callback for publishing sensor data **after the definition row "DHT dht(DHT_PIN, DHT_TYPE);"** with these codes:

```
rcl_publisher_t temp_publisher;

rcl_publisher_t hum_publisher;

sensor_msgs__msg__Temperature temp_msg;

sensor_msgs__msg__RelativeHumidity hum_msg;

rclc_executor_t executor;

rclc_support_t support;

rcl_allocator_t allocator;

rcl_node_t node;

rcl_timer_t timer;


void timer_callback(rcl_timer_t * timer, int64_t last_call_time)

{

  RCLC_UNUSED(last_call_time);

  if (timer == NULL) return;

  float humidity = dht.readHumidity();

  float temperature = dht.readTemperature();

  if (isnan(humidity) || isnan(temperature)) {

    delay(100);

    humidity = dht.readHumidity();

    temperature = dht.readTemperature();

  }

  if (!isnan(humidity) && !isnan(temperature)) {

    temp_msg.temperature = temperature;

    hum_msg.relative_humidity = humidity;

    RCSOFTCHECK(rcl_publish(&temp_publisher, &temp_msg, NULL),

      "publish temp");

    RCSOFTCHECK(rcl_publish(&hum_publisher, &hum_msg, NULL),
```

```
      "publish hum");
  }
}
```

a) temp_publisher, hum_publisher: For publishing temperature and humidity data.
b) temp_msg, hum_msg: ROS message structures for sensor data.
c) executor, support, allocator, node, timer: For micro-ROS infrastructure.
d) Timer_callback: Reads sensor data and retries on invalid readings and publishes valid data to ROS topics using RCSOFTCHECK.

6. Initialize the setup() function by adding the following code that includes micro-ROS initialization.:

```
void setup() {
  Serial.begin(115200);
  dht.begin();
  delay(2000);
  set_microros_transports();
  delay(3000);
  allocator = rcl_get_default_allocator();
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator),
    "support init");
  RCCHECK(rclc_node_init_default(&node, "esp32_node", "", &support),
    "node init");
  RCCHECK(rclc_publisher_init(
    &temp_publisher, &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, Temperature),
    "/dht11_temperature",
    &rmw_qos_profile_sensor_data), "temp publisher");
  RCCHECK(rclc_publisher_init(
    &hum_publisher, &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, RelativeHumidity),
    "/dht11_humidity",
    &rmw_qos_profile_sensor_data), "hum publisher");
  const unsigned int timer_timeout = 2000;
```

```
  RCCHECK(rclc_timer_init_default(&timer, &support,
    RCL_MS_TO_NS(timer_timeout), timer_callback), "timer init");
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator),
    "executor init");
  RCCHECK(rclc_executor_add_timer(&executor, &timer),
    "executor add timer");
  temp_msg.temperature = 0.0;
  hum_msg.relative_humidity = 0.0;
}
```

a) set_microros_transports(): Configures communication by default.
b) rclc_support_init, rclc_node_init_default: Set up the ROS client and node.
c) rclc_publisher_init: Create publishers for /dht11_temperature and /dht11_humidity topics.
d) rclc_timer_init_default: Sets up a timer to call timer_callback every 2 seconds.
e) rclc_executor_init, rclc_executor_add_timer: Manages the timer execution.

7. Implement the loop() function to run the micro-ROS executor.

```
void loop() {
  RCSOFTCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(200)),
    "spin executor");
  delay(200);
}
```

8. Select the port as "/dev/ttyUSB0" from Tools>Port
9. Select the board by Tools>Board>esp32>ESP32 Dev Module
10. Verify and compile the sketch by Sketch>Verify/Compile

11. Open a new terminal:

```
$ ls /dev/tty*

# (After running “ls /dev/tty*”, verify if there is an item named
“/dev/ttyUSB0”)


$ sudo chmod 666 /dev/ttyUSB0

$ sudo usermod -aG dialout $USER

$ newgrp docker


$ groups

# (verify if there is “docker” in the output)
```

12. Upload the sketch by Tools>Upload
13. After uploading is done, open a new terminal and turn on the micro-ROS-agent:

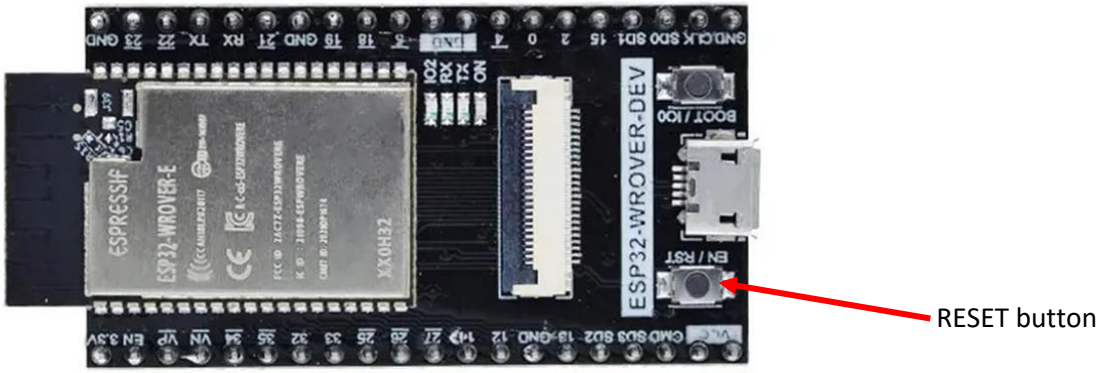
```
$ docker run -it --rm -v /dev:/dev --privileged --net=host microros/micro-
ros-agent:jazzy serial --dev /dev/ttyUSB0 -v6
```

14. Check the message received in the ROS 2 topic by opening a new terminal and running:

```
$ ros2 topic echo /dht11_temperature

$ ros2 topic echo /dht11_humidity
```

a) Temperature and humidity data are published every 2 seconds.
b) Press the RESET button on the ESP32 board if no message is published.



6 Test the MPU9250 sensor on ESP32

What is MPU9250?

The MPU9250 is a 9-axis Motion Processing Unit (MPU) developed by InvenSense. It integrates a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis magnetometer into a single chip, making it ideal for applications requiring motion tracking, orientation, and navigation, such as drones, robotics, and wearable devices. It communicates via I2C or SPI protocols and is commonly used with microcontrollers like Arduino or ESP32.

1. In Arduino IDE, create a new sketch by File>New Sketch
2. Include the necessary libraries and define the pin number of MPU9250's I2C pins and address by adding these codes into the sketch:

```
#include <Wire.h>

#include <MPU9250_asukiaaa.h>

#include <micro_ros_arduino.h>

#include <rcl/rcl.h>

#include <rclc/rclc.h>

#include <rclc/executor.h>

#include <sensor_msgs/msg/imu.h>



#define MPU_SCL 22

#define MPU_SDA 21

#define MPU_ADDRESS 0x68
```

   a) MPU_SCL, MPU_SDA: I2C pins for ESP32.
   b) MPU_ADDRESS: I2C address of the MPU9250 (0x68).

3. Implement macros for error checking in micro-ROS by adding these codes:

```
#define RCCHECK(fn, context) { rcl_ret_t temp_rc = fn; if(temp_rc !=
RCL_RET_OK){return;}}

#define RCSOFTCHECK(fn, context) { rcl_ret_t temp_rc = fn; if(temp_rc !=
RCL_RET_OK){}}
```

4. Implement MPU9250 initialization by adding these codes:

```
MPU9250_asukiaaa mpu(MPU_ADDRESS);
```

5. Define micro-ROS variables and implement the timer callback for publishing sensor data using these codes after **"MPU9250_asukiaaa mpu(MPU_ADDRESS);":**

```
rcl_publisher_t imu_publisher;
sensor_msgs__msg__Imu imu_msg;
rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t timer;

void timer_callback(rcl_timer_t * timer, int64_t last_call_time)
{
  RCLC_UNUSED(last_call_time);
  if (timer == NULL) return;

  // Read and publish MPU9250 data
  if (mpu.accelUpdate() == 0 && mpu.gyroUpdate() == 0) {
    float accel_x = mpu.accelX(); // m/s^2
    float accel_y = mpu.accelY();
    float accel_z = mpu.accelZ();
    float gyro_x = mpu.gyroX() * DEG_TO_RAD; // Convert deg/s to rad/s
    float gyro_y = mpu.gyroY() * DEG_TO_RAD;
    float gyro_z = mpu.gyroZ() * DEG_TO_RAD;
    imu_msg.linear_acceleration.x = accel_x;
    imu_msg.linear_acceleration.y = accel_y;
    imu_msg.linear_acceleration.z = accel_z;
    imu_msg.angular_velocity.x = gyro_x;
    imu_msg.angular_velocity.y = gyro_y;
```

```
    imu_msg.angular_velocity.z = gyro_z;
    // Update timestamp
    unsigned long now = millis();
    imu_msg.header.stamp.sec = now / 1000;
    imu_msg.header.stamp.nanosec = (now % 1000) * 1000000;
    RCSOFTCHECK(rcl_publish(&imu_publisher, &imu_msg, NULL),
      "publish imu");
    delay(50); // I2C recovery delay
  } else {
    mpu.beginAccel(ACC_FULL_SCALE_2_G);
    mpu.beginGyro(GYRO_FULL_SCALE_250_DPS);
    delay(100);
  }
}
```

a) imu_publisher: Publishes IMU data.
b) imu_msg: ROS message for IMU data (acceleration, angular velocity, timestamp).
c) executor, support, allocator, node, timer: micro-ROS infrastructure.
d) Timer_callback:

- Reads accelerometer (m/s²) and gyroscope (converts deg/s to rad/s).
- Sets timestamp using millis().
- Publishes data to the /imu_data topic.
- Retries initialization on failure.

6. Initialize the setup() function by adding the following code that includes micro-ROS initialization:

```
void setup() {
  Serial.begin(115200);
  // Initialize I2C for MPU9250
  Wire.begin(MPU_SDA, MPU_SCL);
  Wire.setClock(100000); // Slower I2C speed for stability
  delay(500); // MPU9250 stabilization

  // Initialize MPU9250
  mpu.setWire(&Wire);
```

```
mpu.beginAccel(ACC_FULL_SCALE_2_G);
mpu.beginGyro(GYRO_FULL_SCALE_250_DPS);
delay(100);
while (1) {
  delay(1000);
  mpu.beginAccel(ACC_FULL_SCALE_2_G);
  mpu.beginGyro(GYRO_FULL_SCALE_250_DPS);
  delay(100);
  if (mpu.accelUpdate() == 0 && mpu.gyroUpdate() == 0) {

    break;
  }
}

// Initialize Micro-ROS
set_microros_transports();
delay(3000); // Increased delay for agent sync

allocator = rcl_get_default_allocator();
RCCHECK(rclc_support_init(&support, 0, NULL, &allocator),
  "support init");
RCCHECK(rclc_node_init_default(&node, "esp32_node", "", &support),
  "node init");

// Initialize publisher
RCCHECK(rclc_publisher_init(
  &imu_publisher, &node,
  ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, Imu),
  "/imu_data",
  &rmw_qos_profile_sensor_data), "imu publisher");

// Initialize timer (2000ms, 0.5Hz)
const unsigned int timer_timeout = 2000;
RCCHECK(rclc_timer_init_default(&timer, &support,
```

```
   RCL_MS_TO_NS(timer_timeout), timer_callback), "timer init");


  // Initialize executor
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator),
    "executor init");
  RCCHECK(rclc_executor_add_timer(&executor, &timer),
    "executor add timer");


  // Initialize message
  imu_msg.header.frame_id.data = (char *)malloc(10 * sizeof(char));
  strcpy(imu_msg.header.frame_id.data, "imu_link");
  imu_msg.header.frame_id.size = strlen(imu_msg.header.frame_id.data);
  imu_msg.header.frame_id.capacity = 10;
  imu_msg.linear_acceleration_covariance[0] = 0.01;
  imu_msg.linear_acceleration_covariance[4] = 0.01;
  imu_msg.linear_acceleration_covariance[8] = 0.01;
  imu_msg.angular_velocity_covariance[0] = 0.01;
  imu_msg.angular_velocity_covariance[4] = 0.01;
  imu_msg.angular_velocity_covariance[8] = 0.01;
}
```

a) set_microros_transports(): Configures communication by default.
b) rclc_support_init, rclc_node_init_default: Set up the ROS client and node.
c) rclc_publisher_init: Create publishers for /imu_data
d) rclc_timer_init_default: Sets up a timer to call timer_callback every 2 seconds.

7. Implement the loop() function to run the micro-ROS executor by adding this code:

```
void loop() {
  RCSOFTCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(200)),
    "spin executor");
  delay(200);
}
```

8. Test the full code to publish MPU9250's data to a ROS 2 topic through micro-ROS.
9. Select the port as "/dev/ttyUSB0" from Tools>Port
10. Select the board by Tools>Board>esp32>ESP32 Dev Module
11. Verify and compile the sketch by Sketch>Verify/Compile
12. Open a new terminal:

```
$ ls /dev/tty*

# (After running "ls /dev/tty*", verify if there is an item named
"/dev/ttyUSB0")

$ sudo chmod 666 /dev/ttyUSB0

$ sudo usermod -aG dialout $USER

$ newgrp docker


$ groups

# (verify if there is "docker" in the output)
```

13. Upload the sketch by Tools>Upload
14. After uploading is done, open a new terminal and turn on the micro-ROS-agent:

```
$ docker run -it --rm -v /dev:/dev --privileged --net=host microros/micro-
ros-agent:jazzy serial --dev /dev/ttyUSB0 -v6
```

15. Check the message received in the ROS 2 topic by opening a new terminal and running:

```
$ source /opt/ros/jazzy/setup.bash

$ ros2 topic echo /imu_data
```

Acceleration (m/s^2) and gyro(rad/s) data are published every 2 seconds.

7 Create a control program for the DHT11 sensor and the MPU9250 sensor

Integrating the testing programs of DHT11 and MPU9250, a control program can publish temperature, humidity, acceleration and gyroscope data to different ROS2 topics.

1. In Arduino IDE, create a new sketch by File>New Sketch

2. Setting up libraries and pin assignments by adding the following code into the sketch:

```
#include <DHT.h>

#include <Wire.h>

#include <MPU9250_asukiaaa.h>

#include <micro_ros_arduino.h>

#include <rcl/rcl.h>

#include <rclc/rclc.h>

#include <rclc/executor.h>

#include <sensor_msgs/msg/temperature.h>

#include <sensor_msgs/msg/relative_humidity.h>

#include <sensor_msgs/msg/imu.h>


#define DHT_PIN 27

#define DHT_TYPE DHT11

#define MPU_SCL 22

#define MPU_SDA 21

#define MPU_ADDRESS 0x68
```

3. Implement macros for error checking in micro-ROS by adding these codes:

```
#define RCCHECK(fn, context) { rcl_ret_t temp_rc = fn; if(temp_rc !=
RCL_RET_OK){return;}}

#define RCSOFTCHECK(fn, context) { rcl_ret_t temp_rc = fn; if(temp_rc !=
RCL_RET_OK){}}
```

4. Initialize the DHT11 and MPU9250 sensors' setup function by adding these codes:

```
DHT dht(DHT_PIN, DHT_TYPE);

MPU9250_asukiaaa mpu(MPU_ADDRESS);
```

5. Define micro-ROS variables and implement the timer callback for publishing sensor data using these codes after **"MPU9250_asukiaaa mpu(MPU_ADDRESS);":**

```
rcl_publisher_t temp_publisher;
rcl_publisher_t hum_publisher;
rcl_publisher_t imu_publisher;
sensor_msgs__msg__Temperature temp_msg;
sensor_msgs__msg__RelativeHumidity hum_msg;
sensor_msgs__msg__Imu imu_msg;
rclc_executor_t executor;
rclc_support_t support;
rcl_allocator_t allocator;
rcl_node_t node;
rcl_timer_t timer;

void timer_callback(rcl_timer_t * timer, int64_t last_call_time)
{
  RCLC_UNUSED(last_call_time);
  if (timer == NULL) return;

  // Read and publish DHT11 data
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();
  if (isnan(humidity) || isnan(temperature)) {
    delay(100);
    humidity = dht.readHumidity();
    temperature = dht.readTemperature();
  }
  if (!isnan(humidity) && !isnan(temperature)) {
    temp_msg.temperature = temperature;
    hum_msg.relative_humidity = humidity;
    RCSOFTCHECK(rcl_publish(&temp_publisher, &temp_msg, NULL),
      "publish temp");
    RCSOFTCHECK(rcl_publish(&hum_publisher, &hum_msg, NULL),
```

```
      "publish hum");
  }

  // Read and publish MPU9250 data
  if (mpu.accelUpdate() == 0 && mpu.gyroUpdate() == 0) {
    float accel_x = mpu.accelX(); // m/s^2
    float accel_y = mpu.accelY();
    float accel_z = mpu.accelZ();
    float gyro_x = mpu.gyroX() * DEG_TO_RAD; // Convert deg/s to rad/s
    float gyro_y = mpu.gyroY() * DEG_TO_RAD;
    float gyro_z = mpu.gyroZ() * DEG_TO_RAD;
    imu_msg.linear_acceleration.x = accel_x;
    imu_msg.linear_acceleration.y = accel_y;
    imu_msg.linear_acceleration.z = accel_z;
    imu_msg.angular_velocity.x = gyro_x;
    imu_msg.angular_velocity.y = gyro_y;
    imu_msg.angular_velocity.z = gyro_z;
    // Update timestamp
    unsigned long now = millis();
    imu_msg.header.stamp.sec = now / 1000;
    imu_msg.header.stamp.nanosec = (now % 1000) * 1000000;
    RCSOFTCHECK(rcl_publish(&imu_publisher, &imu_msg, NULL),
      "publish imu");
    delay(50); // I2C recovery delay
  } else {
    mpu.beginAccel(ACC_FULL_SCALE_2_G);
    mpu.beginGyro(GYRO_FULL_SCALE_250_DPS);
    delay(100);
  }
}
```

6. Implement the loop function to run the micro-ROS executor.

```
void setup() {
  Serial.begin(115200);
  // Initialize I2C for MPU9250
  Wire.begin(MPU_SDA, MPU_SCL);
  Wire.setClock(100000); // Slower I2C speed for stability
  delay(500); // MPU9250 stabilization

  // Test DHT11
  dht.begin();
  delay(2000); // DHT11 stabilization

  // Initialize MPU9250
  mpu.setWire(&Wire);
  mpu.beginAccel(ACC_FULL_SCALE_2_G);
  mpu.beginGyro(GYRO_FULL_SCALE_250_DPS);
  delay(100);
  while (1) {
    delay(1000);
    mpu.beginAccel(ACC_FULL_SCALE_2_G);
    mpu.beginGyro(GYRO_FULL_SCALE_250_DPS);
    delay(100);
    if (mpu.accelUpdate() == 0 && mpu.gyroUpdate() == 0) {
      break;
    }
  }

  // Initialize Micro-ROS
  set_microros_transports();
  delay(3000); // Increased delay for agent sync

  allocator = rcl_get_default_allocator();
  RCCHECK(rclc_support_init(&support, 0, NULL, &allocator),
    "support init");
```

```
  RCCHECK(rclc_node_init_default(&node, "esp32_node", "", &support),
    "node init");

  // Initialize publishers
  RCCHECK(rclc_publisher_init(
    &temp_publisher, &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, Temperature),
    "/dht11_temperature",
    &rmw_qos_profile_sensor_data), "temp publisher");
  RCCHECK(rclc_publisher_init(
    &hum_publisher, &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, RelativeHumidity),
    "/dht11_humidity",
    &rmw_qos_profile_sensor_data), "hum publisher");
  RCCHECK(rclc_publisher_init(
    &imu_publisher, &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(sensor_msgs, msg, Imu),
    "/imu_data",
    &rmw_qos_profile_sensor_data), "imu publisher");

  // Initialize timer (2000ms, 0.5Hz)
  const unsigned int timer_timeout = 2000;
  RCCHECK(rclc_timer_init_default(&timer, &support,
RCL_MS_TO_NS(timer_timeout), timer_callback), "timer init");

  // Initialize executor
  RCCHECK(rclc_executor_init(&executor, &support.context, 1, &allocator),
    "executor init");
  RCCHECK(rclc_executor_add_timer(&executor, &timer),
    "executor add timer");
  // Initialize messages
  temp_msg.temperature = 0.0;
  hum_msg.relative_humidity = 0.0;
  imu_msg.header.frame_id.data = (char *)malloc(10 * sizeof(char));
```

```
  strcpy(imu_msg.header.frame_id.data, "imu_link");
  imu_msg.header.frame_id.size = strlen(imu_msg.header.frame_id.data);
  imu_msg.header.frame_id.capacity = 10;
  imu_msg.linear_acceleration_covariance[0] = 0.01;
  imu_msg.linear_acceleration_covariance[4] = 0.01;
  imu_msg.linear_acceleration_covariance[8] = 0.01;
  imu_msg.angular_velocity_covariance[0] = 0.01;
  imu_msg.angular_velocity_covariance[4] = 0.01;
  imu_msg.angular_velocity_covariance[8] = 0.01;
}
```

7. Implement the loop function to run the micro-ROS executor.

```
void loop() {
  RCSOFTCHECK(rclc_executor_spin_some(&executor, RCL_MS_TO_NS(200)),
    "spin executor");
  delay(200);
}
```

8. Configure the Serial Port by opening a new terminal and running:

```
$ ls /dev/tty*
# (After running “ls /dev/tty*”, verify if there is an item named
“/dev/ttyUSB0”)
$ sudo chmod 666 /dev/ttyUSB0
$ sudo usermod -aG dialout $USER
$ newgrp docker


$ groups
# (verify if there is “docker” in the output)
```

9. Set Tools > Board > esp32> ESP32 Dev Module, Port > /dev/ttyUSB0.
10. Go to Sketch > Verify/Compile, if successfully compiled, Sketch > Upload.

11. When the full code is uploaded successfully, open a new terminal and run micro-ROS Agent:

```
$ docker run -it --rm -v /dev:/dev --privileged --net=host microros/micro-ros-agent:jazzy serial --dev /dev/ttyUSB0 -v6
```

12. List Topics

```
$ source /opt/ros/jazzy/setup.bash

$ ros2 topic list
```

13. Monitor Topics:

```
$ ros2 topic echo /dht11_temperature

$ ros2 topic echo /dht11_humidity

$ ros2 topic echo /imu_data
```

6 Data Visualisation by a Python Subscriber

1. Install matplotlib

```
$ sudo apt update -y

$ sudo apt install python3-matplotlib


$ python3 -c "import matplotlib; print(matplotlib.__version__)"
# (Verify if the matplotlib is installed)
```

2. Create a new file “plot_dht11_data.py” in the VS Code in the path “microros_ws” and add these codes to it.

```
import rclpy

from rclpy.node import Node

from rclpy.qos import QoSProfile, QoSReliabilityPolicy, QoSHistoryPolicy

from sensor_msgs.msg import Temperature, RelativeHumidity

import matplotlib.pyplot as plt

import time


class DHT11Subscriber(Node):

    def __init__(self):

        super().__init__('dht11_subscriber')
```

```
        # Use QoS profile matching sensor_data
        qos = QoSProfile(
            reliability=QoSReliabilityPolicy.BEST_EFFORT,
            history=QoSHistoryPolicy.KEEP_LAST,
            depth=7
        )
        self.temp_data = []
        self.hum_data = []
        self.start_time = time.time()

        # Subscribers
        self.temp_sub = self.create_subscription(
            Temperature, '/dht11_temperature', self.temp_callback, qos)
        self.hum_sub = self.create_subscription(
            RelativeHumidity, '/dht11_humidity', self.hum_callback, qos)

        # Timer to stop after 2 minutes
        self.timer = self.create_timer(1.0, self.check_duration)

    def temp_callback(self, msg):
        elapsed_time = time.time() - self.start_time
        if elapsed_time <= 120.0:  # Collect for 2 minutes
            self.temp_data.append((elapsed_time, msg.temperature))
            self.get_logger().info(
f'Received Temperature: {msg.temperature} C at {elapsed_time:.2f} s')

    def hum_callback(self, msg):
        elapsed_time = time.time() - self.start_time
        if elapsed_time <= 120.0:
            self.hum_data.append((elapsed_time, msg.relative_humidity))
            self.get_logger().info(
f'Received Humidity: {msg.relative_humidity} % at {elapsed_time:.2f} s')

    def check_duration(self):
```

```
        elapsed_time = time.time() - self.start_time
        if elapsed_time > 120.0:
            self.get_logger().info(
               '2 minutes elapsed, plotting DHT11 data...')
            self.timer.cancel()
            self.plot_data()
            self.destroy_node()
            rclpy.shutdown()


    def plot_data(self):
        # Plot Temperature
        plt.figure(figsize=(10, 6))
        if self.temp_data:
            times, values = zip(*self.temp_data)
            plt.plot(times, values, label='Temperature (C)')
            plt.xlabel('Time (s)')
            plt.ylabel('Temperature (C)')
            plt.title('Temperature over 2 Minutes')
            plt.legend()
            plt.grid(True)
            # plt.show()
        else:
            self.get_logger().warn('No temperature data received')


        # Plot Humidity
        plt.figure(figsize=(10, 6))
        if self.hum_data:
            times, values = zip(*self.hum_data)
            plt.plot(times, values, label='Humidity (%)')
            plt.xlabel('Time (s)')
            plt.ylabel('Humidity (%)')
            plt.title('Humidity over 2 Minutes')
            plt.legend()
            plt.grid(True)
```

```
            plt.show()
        else:
            self.get_logger().warn('No humidity data received')


def main():
    rclpy.init()
    node = DHT11Subscriber()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        node.get_logger().info('Interrupted, shutting down...')
    finally:
        node.destroy_node()
        rclpy.shutdown()


if __name__ == '__main__':
    main()
```

3. Ensure the micro-ROS Agent is running. If not, upload the control program to the ESP32 board and run this command in a new terminal to start the agent.

```
$ docker run -it --rm -v /dev:/dev --privileged --net=host microros/micro-ros-agent:jazzy serial --dev /dev/ttyUSB0 -v6
```

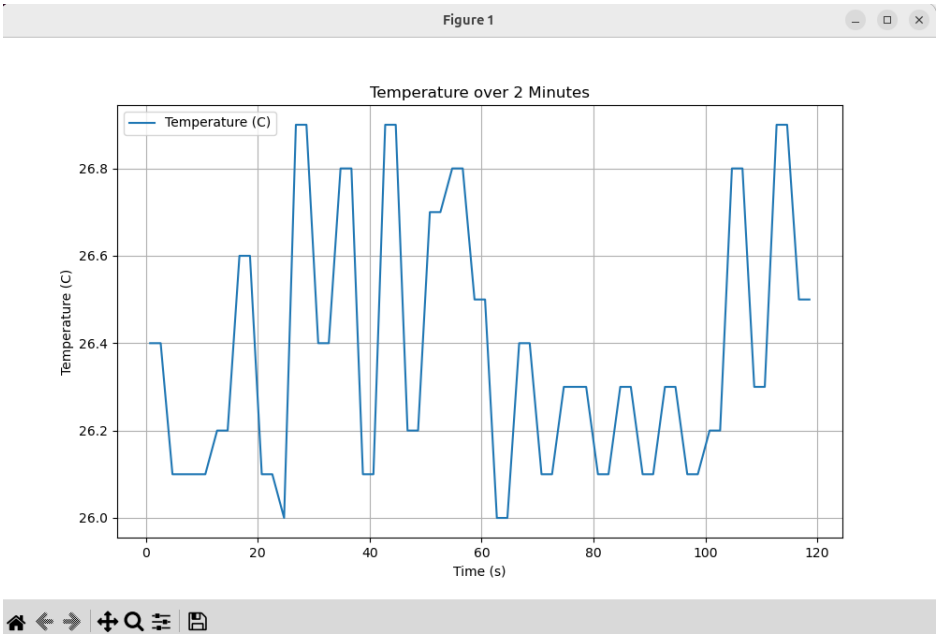
4. Confirm the topics are working properly. Have a trial on /dht11_temperature

```
$ source /opt/ros/jazzy/setup.bash
$ ros2 topic echo /dht11_temperature
```

5. Run the plot_dht11_data.py. Open a new terminal and run the script.

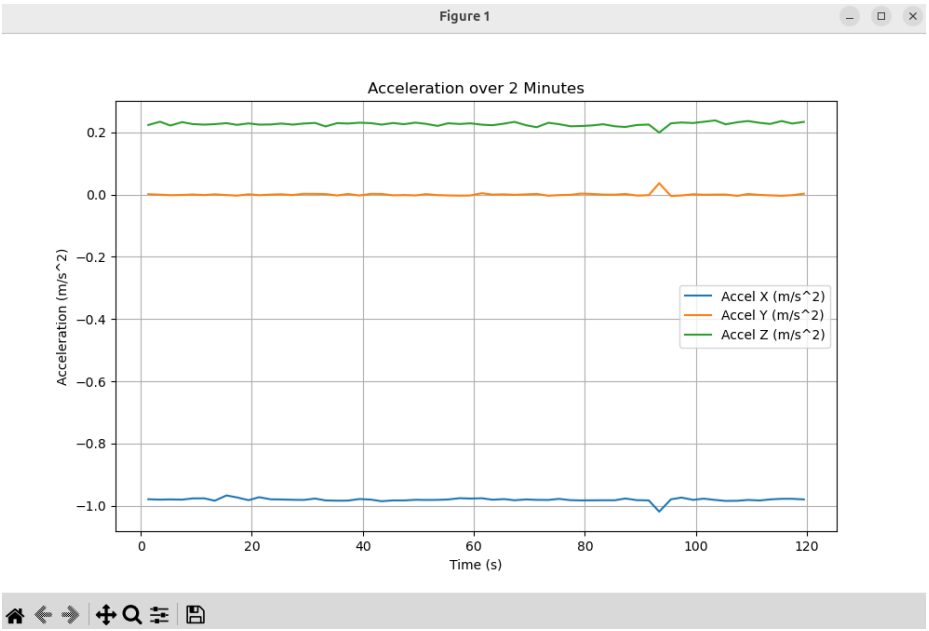
```
$ source /opt/ros/jazzy/setup.bash
$ python3 ~/microros_ws/plot_dht11_data.py
```

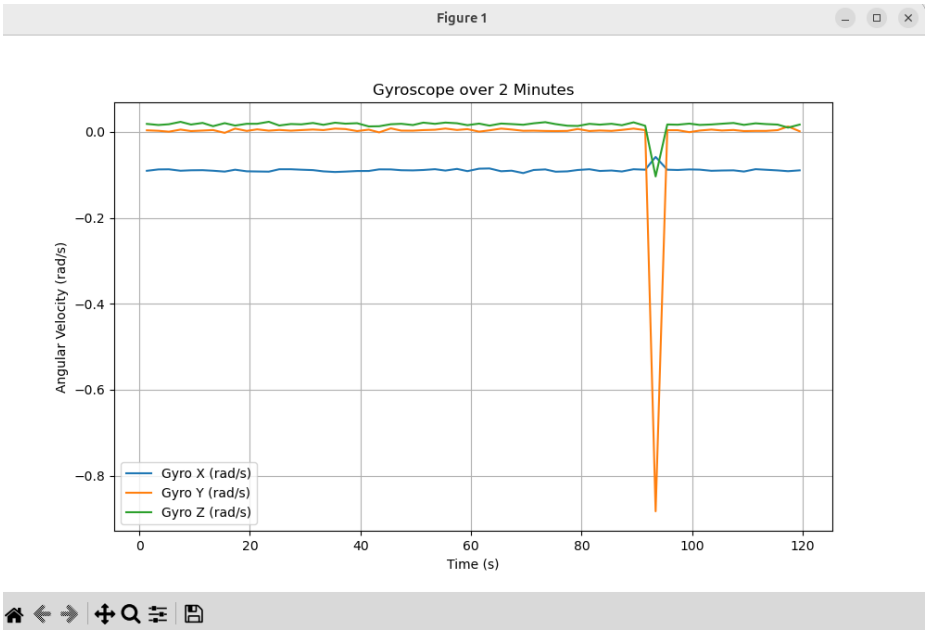
If the script worked normally, it would record the temperature and humidity data for the next 2 minutes and plot it.



8 Exercise

Try to write a Python script to read the MPU9250's acceleration and gyroscope data.





9 Reference

https://github.com/micro-ROS/micro_ros_arduino