Robotics Hardware Training : Practical Lab Guide

Folder layout (one workspace for everything)

Create a ros2 workspace (example uses `ros2_ws`) :

```
~/ros2_ws/
  src/
    xm/                        # your package
      package.xml
      setup.py
      xm/
        __init__.py
        simple_serial_node.py
        imu_serial_node.py
        imu_complementary_node.py
        lidar_republisher.py
```

`colcon build` will install nodes into `install/`.

Example 1 : Send a simple message from Arduino → PC (ROS2)

Goal: Send `"Hello ROS2 from Arduino"` over serial and publish as `/chatter` (String).

Arduino code (upload to Arduino)

File: `arduino_hello.ino`

```
void setup() {
  Serial.begin(115200);
  delay(1000);
}
```

```
void loop() {
  Serial.println("Hello ROS2 from Arduino");
  delay(1000);
}
```

ROS2 Python node (reads serial & publishes String)

File: `xm/simple_serial_node.py`

```
# xm/simple_serial_node.py
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
import serial

class SimpleSerialNode(Node):
    def __init__(self):
        super().__init__('simple_serial_node')
        self.pub = self.create_publisher(String, 'chatter', 10)
        self.ser = serial.Serial('/dev/ttyUSB0', 115200, timeout=1)
        self.timer = self.create_timer(1.0, self.read_serial)

    def read_serial(self):
        if self.ser.in_waiting:
            line = self.ser.readline().decode('utf-8',
errors='ignore').strip()
            if line:
                msg = String()
                msg.data = line
                self.pub.publish(msg)
                self.get_logger().info(f"Published: {line}")

def main(args=None):
    rclpy.init(args=args)
    node = SimpleSerialNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()
```

Build & run

1. Put code in package folder `xm/xm/simple_serial_node.py`.
2. Add entry point in `setup.py` (example below).
3. Build:

```
cd ~/ros2_ws
colcon build
source install/setup.bash
ros2 run xm simple_serial_node
```

4. In another terminal:

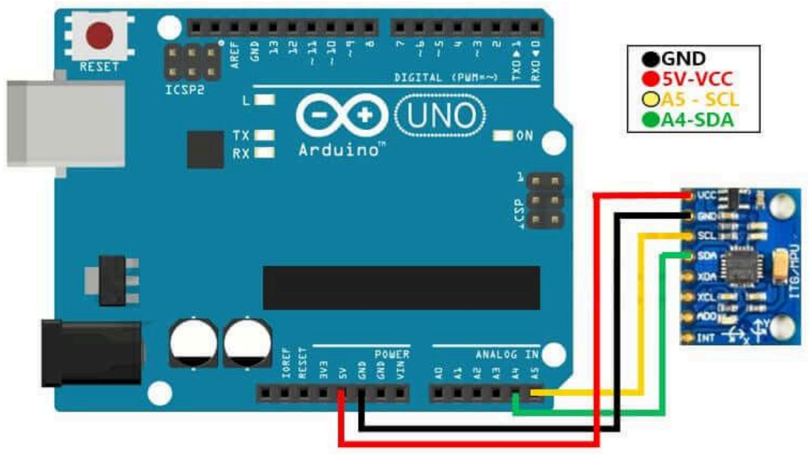
```
ros2 topic echo /chatter
```

Schematic :

```
Arduino (USB) ---> PC USB ---> simple_serial_node -> publishes
/chatter -> ros2 topic echo
```

Example 2 : IMU: Arduino reads MPU-6050 → PC → ROS2 → RViz (orientation + TF)

Goal: Read IMU on Arduino, send CSV over serial, fuse with complementary filter in ROS2, publish `sensor_msgs/Imu` with orientation and broadcast TF so RViz can show axes.



2.A Arduino code (MPU-6050) — CSV format

File: `arduino_imu.ino`

```
#include <Wire.h>
const int MPU_ADDR = 0x68;
const float ACCEL_SCALE = 16384.0;
const float GYRO_SCALE = 131.0;

void setup() {
  Wire.begin();
  Serial.begin(115200);
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x6B); Wire.write(0); Wire.endTransmission(true);
  // configure ±2g and ±250°/s if needed
}

void loop() {
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_ADDR, 14, true);

  float accX = (Wire.read() << 8 | Wire.read()) / ACCEL_SCALE;
  float accY = (Wire.read() << 8 | Wire.read()) / ACCEL_SCALE;
  float accZ = (Wire.read() << 8 | Wire.read()) / ACCEL_SCALE;
  Wire.read(); Wire.read(); // temp
  float gyroX = (Wire.read() << 8 | Wire.read()) / GYRO_SCALE;
  float gyroY = (Wire.read() << 8 | Wire.read()) / GYRO_SCALE;
  float gyroZ = (Wire.read() << 8 | Wire.read()) / GYRO_SCALE;

  // send as CSV: ax,ay,az,gx,gy,gz
  Serial.print(accX, 3); Serial.print(',');
  Serial.print(accY, 3); Serial.print(',');
  Serial.print(accZ, 3); Serial.print(',');
  Serial.print(gyroX, 2); Serial.print(',');
  Serial.print(gyroY, 2); Serial.print(',');
  Serial.println(gyroZ, 2);
  delay(10); // ~100 Hz
}
```

2.B ROS2 node — complementary filter + TF broadcast

File: `xm/imu_complementary_node.py`

```
# xm/imu_complementary_node.py
import rclpy, math
from rclpy.node import Node
from sensor_msgs.msg import Imu
from tf2_ros import TransformBroadcaster
from geometry_msgs.msg import TransformStamped
import serial

class IMUComplementaryNode(Node):
    def __init__(self):
        super().__init__('imu_serial_node')
        self.pub = self.create_publisher(Imu, 'imu/data_raw', 10)
        self.tf_broad = TransformBroadcaster(self)
        self.ser = serial.Serial('/dev/ttyUSB0', 115200, timeout=1)
        self.ser.reset_input_buffer()
        self.alpha = 0.98
        self.dt = 0.01
        self.roll = 0.0
        self.pitch = 0.0
        self.yaw = 0.0
        self.create_timer(self.dt, self.read_serial)

    def read_serial(self):
        if self.ser.in_waiting:
            line = self.ser.readline().decode('utf-8',
errors='ignore').strip()
            parts = line.split(',')
            if len(parts) >= 6:
                try:
                    ax, ay, az = map(float, parts[0:3])
                    gx, gy, gz = map(float, parts[3:6])
                except ValueError:
                    return

                # accel -> roll/pitch
                roll_acc = math.atan2(ay, az)
                pitch_acc = math.atan(-ax / math.sqrt(ay*ay +
az*az))
```

```
                # gyro deg/s -> rad/s
                gx = math.radians(gx); gy = math.radians(gy); gz =
math.radians(gz)

                # complementary filter
                self.roll = self.alpha * (self.roll + gx * self.dt)
+ (1 - self.alpha) * roll_acc
                self.pitch = self.alpha * (self.pitch + gy *
self.dt) + (1 - self.alpha) * pitch_acc
                self.yaw += gz * self.dt

                # quaternion
                cy = math.cos(self.yaw * 0.5); sy =
math.sin(self.yaw * 0.5)
                cp = math.cos(self.pitch * 0.5); sp =
math.sin(self.pitch * 0.5)
                cr = math.cos(self.roll * 0.5); sr =
math.sin(self.roll * 0.5)
                qx = sr*cp*cy - cr*sp*sy
                qy = cr*sp*cy + sr*cp*sy
                qz = cr*cp*sy - sr*sp*cy
                qw = cr*cp*cy + sr*sp*sy

                # publish Imu
                msg = Imu()
                msg.header.stamp = self.get_clock().now().to_msg()
                msg.header.frame_id = 'imu_link'
                msg.orientation.x = qx; msg.orientation.y = qy;
msg.orientation.z = qz; msg.orientation.w = qw
                msg.orientation_covariance[0] = 0.001
                msg.angular_velocity.x = gx; msg.angular_velocity.y
= gy; msg.angular_velocity.z = gz
                msg.linear_acceleration.x = ax*9.81;
msg.linear_acceleration.y = ay*9.81; msg.linear_acceleration.z =
az*9.81
                self.pub.publish(msg)

                # broadcast TF (imu_link relative to base_link)
                t = TransformStamped()
                t.header.stamp = msg.header.stamp
                t.header.frame_id = 'base_link'
                t.child_frame_id = 'imu_link'
                t.transform.rotation.x = qx; t.transform.rotation.y
= qy; t.transform.rotation.z = qz; t.transform.rotation.w = qw
                t.transform.translation.x = 0.0;
t.transform.translation.y = 0.0; t.transform.translation.z = 0.0
                self.tf_broad.sendTransform(t)
        # else: nothing to read

def main(args=None):
    rclpy.init(args=args)
    node = IMUComplementaryNode()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    node.destroy_node()
    rclpy.shutdown()
```

Build & run

```
colcon build
source install/setup.bash
ros2 run xm imu_complementary_node
```

RViz setup

1. Open `rviz2`.
2. Set Fixed Frame = `base_link`.
3. Add `TF` display → you should see `imu_link` frame rotating.
4. Optional: add `Axes` and set Frame = `imu_link` to show local axes.

IMU schematic

```
MPU-6050 (I2C: SDA,SCL)
   |
Arduino (reads RAW accel+gyro)
   |
USB Serial
   |
PC -- imu_complementary_node --> /imu/data_raw +
TF(base_link->imu_link)
   |
RViz (TF / Axes)
```

Example 3 : LiDAR: use `rplidar_ros` driver → publish `/scan` → RViz

Goal: Use official ROS2 driver to publish LiDAR scans; optionally re-publish with a Python node.

3.A Install & launch official driver

(assuming ROS2 Humble)

```
sudo apt update
sudo apt install ros-humble-rplidar-ros
```

Launch (use correct launch file — for A1 variants):

```
ros2 launch rplidar_ros view_rplidar_a1_launch.py
serial_port:=/dev/ttyUSB0 frame_id:=laser serial_baudrate:=115200
```

> If your device/bundle requires `serial_baudrate:=256000`, change accordingly.

3.B Python wrapper (re-publisher) — optional

File: `xm/lidar_republisher.py`

```
# xm/lidar_republisher.py
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import LaserScan

class LidarRepublisher(Node):
    def __init__(self):
        super().__init__('lidar_republisher')
        self.sub = self.create_subscription(LaserScan, '/scan',
self.cb_scan, 10)
        self.pub = self.create_publisher(LaserScan, '/my_scan', 10)

    def cb_scan(self, msg):
        # Example: passthrough (you can filter ranges here)
        self.pub.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    node = LidarRepublisher()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()
```

Run:

```
ros2 run xm lidar_republisher
```

Then in RViz: add `LaserScan` topic `/my_scan` (or `/scan` directly if not re-publishing), fixed frame `laser`.

LiDAR schematic

```
RPLIDAR USB -> PC -> rplidar_ros driver -> /scan -> RViz LaserScan
(optional) -> lidar_republisher -> /my_scan
```

Example 4 : Running IMU ROS2 Node on Raspberry Pi Through SSH

Objective

Use a Raspberry Pi as the compute unit:

- Arduino → sends IMU data
- Raspberry Pi → receives data via USB
- Raspberry Pi → runs ROS2 IMU node
- PC → connects via SSH only
- PC does not run any ROS2 nodes

Hardware Connections

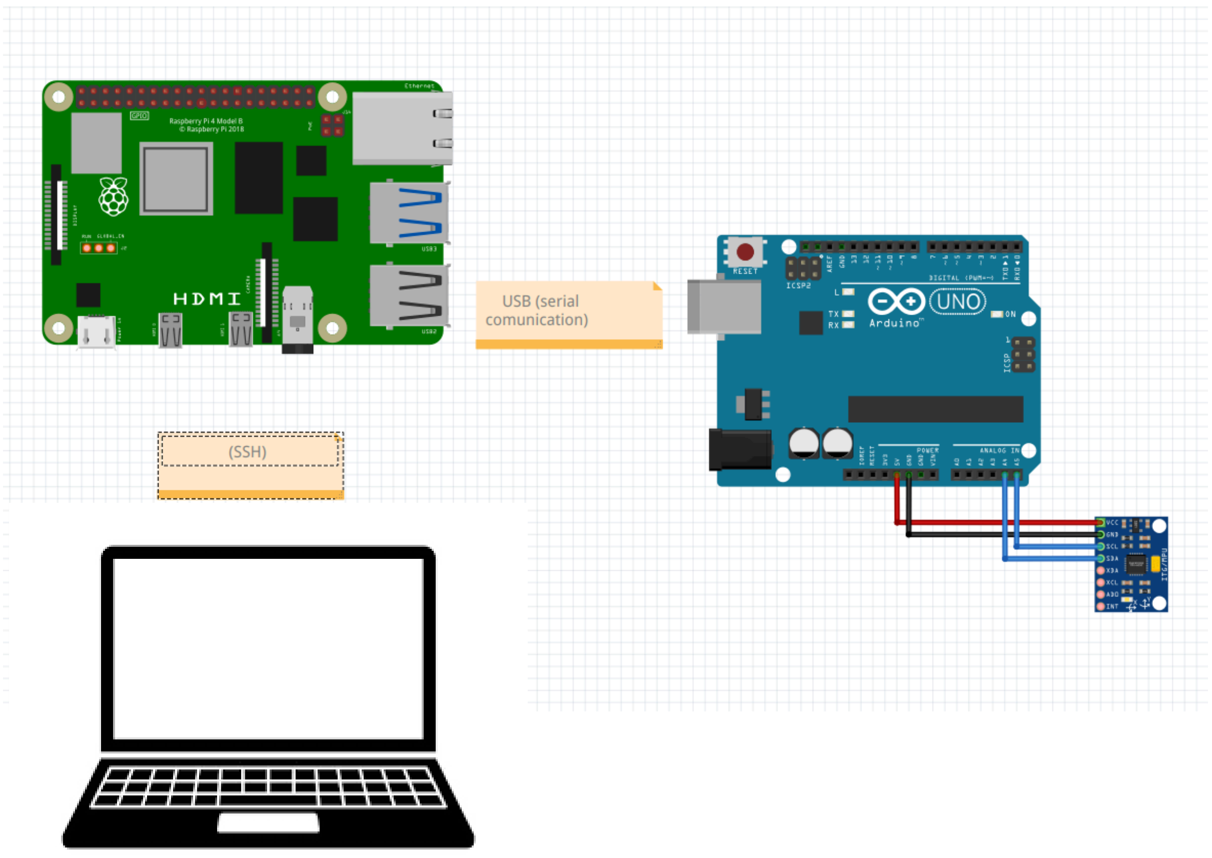
Arduino connects directly to Raspberry Pi USB port

```
[IMU Sensor] → Arduino → USB cable → Raspberry Pi
```

Your laptop only connects via SSH:

```
Laptop (VSCode SSH) → Raspberry Pi
```

No ROS networking between laptop and Raspberry Pi.



Step 1 : SSH Into Raspberry Pi

On your laptop:

```
ssh pi@192.168.1.20
```

Or use VSCode → Remote SSH.

Complete Guide — Install Ubuntu on Raspberry Pi, Connect via SSH, and Use VS Code

1 — Choose the Correct Ubuntu Image

Architecture:
Use **64-bit (arm64)** for Raspberry Pi 4.

Recommended version:
Ubuntu Server 22.04 LTS (64-bit, arm64)

Do not use:

- Ubuntu Desktop (too heavy)
- Ubuntu Core (IoT version, not suitable)
- Ubuntu Server armhf (32-bit, outdated)

---

2 — Download the Ubuntu Image

Option A: Using Raspberry Pi Imager
Download it from the Raspberry Pi website.
In the Imager, select:
Operating System → Other general-purpose OS → Ubuntu → Ubuntu Server 22.04 LTS (64-bit)

Option B: Manual Download
Go to the Ubuntu website → Raspberry Pi section → Download **Ubuntu Server 22.04 ARM64**.

---

3 — Flash the SD Card

Use Raspberry Pi Imager.

Steps:

1. Open Raspberry Pi Imager.
2. Choose Device → Raspberry Pi 4.
3. Choose OS → Ubuntu Server 22.04 LTS (64-bit).
4. Choose Storage → select the SD card.
5. Open Advanced Options.
6. Configure the following:

Required settings:

- Hostname: ubuntu
- Enable SSH: Yes

- Username: ubuntu
- Password: ubuntu

Optional settings (if using Wi-Fi):

- Wi-Fi SSID: your Wi-Fi name
- Wi-Fi Password: your password
- Wi-Fi Country: your country

Save and click Write.

This creates an SD card with SSH enabled and optional Wi-Fi preconfigured.

---

4 — Boot the Raspberry Pi

Insert the SD card into the Raspberry Pi and power it.
If possible, plug in an Ethernet cable for guaranteed connectivity.
If using Wi-Fi, ensure the SSID and password are correct.
First boot usually takes 20–40 seconds.

---

5 — Find the Raspberry Pi IP Address

Method 1: Using nmap

```
nmap -sn 192.168.1.0/24
```

Method 2: Using arp

```
arp -a
```

Method 3: Using your router
Open your router at 192.168.1.1, then check DHCP Client List.
Look for a device named "ubuntu" or identified as Raspberry Pi.

---

6 — Connect to the Raspberry Pi via SSH

Example: if the Raspberry Pi’s IP is 192.168.1.25

```
ssh ubuntu@192.168.1.25
```

Default password: ubuntu
On first login, the system will require you to change the password.

Connect Raspberry Pi to VS Code Using SSH

7 — Install the Remote SSH Extension

In VS Code:

- Open the Extensions panel (Ctrl+Shift+X).
- Search for "Remote - SSH" by Microsoft.
- Install it.

---

8 — Connect to the Raspberry Pi

Press Ctrl+Shift+P and choose:
Remote-SSH: Connect to Host…

You will see your hosts (from ~/.ssh/config), for example:

- 192.168.1.25

Select the Raspberry Pi.

---

9 — Accept the Connection and Enter Password

On first connection:

- Accept the host key (type "yes").
- Enter the Raspberry Pi password.

VS Code will open a new remote window connected to the Raspberry Pi.

---

10 — Open Your ROS 2 Workspace

In the remote VS Code window:
File → Open Folder
Navigate to:

```
/home/ubuntu/ros2_ws
```

Open the folder.

You can now edit files, run ROS2 commands, create packages, and develop directly on the Raspberry Pi from your PC

Step 2 : Open ROS2 Workspace on Raspberry Pi

Inside Raspberry Pi:

```
ros2_ws/
 ├── src/
 │     └── imu_reader/
 ├── build/
 ├── install/
 └── log/
```

You edit code from laptop VSCode,
 but files live **on the Raspberry Pi**.

Step 3 : Install Required Packages on Raspberry Pi

```
sudo apt install ros-humble-rclpy ros-humble-sensor-msgs
ros-humble-tf2-ros python3-serial
```

Step 4 : Add the IMU Node Package

Inside workspace:

```
cd ~/ros2_ws/src
ros2 pkg create imu_reader --build-type ament_python
```

Save inside:

```
ros2_ws/src/imu_reader/imu_reader/imu_serial_node.py
```

```
import rclpy, math
from rclpy.node import Node
from sensor_msgs.msg import Imu
from tf2_ros import TransformBroadcaster
from geometry_msgs.msg import TransformStamped
import serial

class IMUSerialNode(Node):
    def __init__(self):
        super().__init__('imu_serial_node')

        self.pub = self.create_publisher(Imu, 'imu/data_raw', 10)
        self.tf_broad = TransformBroadcaster(self)

        self.ser = serial.Serial('/dev/ttyUSB0', 115200, timeout=1)
        self.ser.reset_input_buffer()

        self.alpha = 0.98
        self.dt = 0.01
        self.roll = 0.0
        self.pitch = 0.0
        self.yaw = 0.0

        self.create_timer(self.dt, self.read_serial)

    def read_serial(self):
        if not self.ser.in_waiting:
            return

        line = self.ser.readline().decode().strip()
        parts = line.split(',')
        if len(parts) < 6:
            return

        try:
            ax, ay, az = map(float, parts[0:3])
            gx, gy, gz = map(float, parts[3:6])
        except:
            return

        roll_acc = math.atan2(ay, az)
        pitch_acc = math.atan(-ax / math.sqrt(ay*ay + az*az))

        gx = math.radians(gx); gy = math.radians(gy); gz =
math.radians(gz)

        self.roll = self.alpha*(self.roll + gx*self.dt) +
(1-self.alpha)*roll_acc
        self.pitch = self.alpha*(self.pitch + gy*self.dt) +
(1-self.alpha)*pitch_acc
        self.yaw += gz*self.dt

        cy = math.cos(self.yaw*0.5); sy = math.sin(self.yaw*0.5)
        cp = math.cos(self.pitch*0.5); sp = math.sin(self.pitch*0.5)
        cr = math.cos(self.roll*0.5); sr = math.sin(self.roll*0.5)

        qx = sr*cp*cy - cr*sp*sy
        qy = cr*sp*cy + sr*cp*sy
        qz = cr*cp*sy - sr*sp*cy
        qw = cr*cp*cy + sr*sp*sy

        msg = Imu()
        msg.header.stamp = self.get_clock().now().to_msg()
        msg.header.frame_id = 'imu_link'

        msg.orientation.x = qx
        msg.orientation.y = qy
        msg.orientation.z = qz
        msg.orientation.w = qw

        msg.angular_velocity.x = gx
        msg.angular_velocity.y = gy
        msg.angular_velocity.z = gz

        msg.linear_acceleration.x = ax*9.81
        msg.linear_acceleration.y = ay*9.81
        msg.linear_acceleration.z = az*9.81

        self.pub.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    node = IMUSerialNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()
```

Step 5 : Build on Raspberry Pi

```
cd ~/ros2_ws
colcon build
source install/setup.bash
```

Step 6 : Run the Node (on Raspberry Pi)

```
ros2 run imu_reader imu_serial_node
```