🌤️ Build Your First Robot: Smart Weather Station

ELIF10 Tutorial: Explain Like I'm 10!

Learn robotics by building a weather monitoring robot that collects temperature, humidity, and light data. This tutorial shows **EVERY feature** of RobotStudio from start to deployment!

🎯 What You'll Build

A **Weather Station Robot** that:

- ✓ Measures temperature and humidity (DHT22 sensor)
- ✓ Measures light levels (photoresistor)
- ✓ Displays data in real-time 3D visualization
- ✓ Publishes data to ROS2 topics
- ✓ Logs data to files
- ✓ Has a blinking status LED
- ✓ Runs on Arduino or ESP32

Why this project?

- Simple hardware (under $20)
- Teaches ROS2 fundamentals
- Shows ALL RobotStudio features
- Real-world application
- Easy to expand later

🛒 Shopping List (Under $20!)

Option A: Arduino-based ($15)

- Arduino Uno or Nano ($8)
- DHT22 Temperature/Humidity Sensor ($5)
- Photoresistor (light sensor) ($1)
- LED + Resistor ($1)
- Breadboard + Jumper Wires ($3)

Option B: ESP32-based ($12)

- ESP32 Dev Board ($6) - **Recommended! Has WiFi**
- DHT22 Sensor ($5)
- Photoresistor ($1)
- Breadboard + Wires ($3)

Where to Buy

- Amazon: "Arduino starter kit" or "ESP32 kit"
- AliExpress: Cheaper but slower shipping
- Local electronics store

Don't have hardware yet? No problem! Follow along with the **simulation mode** (no hardware required).

📚 What You'll Learn

By the end of this tutorial, you'll know how to use:

RobotStudio Features (ALL OF THEM!)

1. ✓ **Workflow Wizard** - 5-minute robot creation
2. ✓ **Visual URDF Editor** - Design robot structure
3. ✓ **Node Graph Editor** - Program robot behavior
4. ✓ **Code Generation** - Auto-generate ROS2 Python code
5. ✓ **MuJoCo Simulation** - Test in 3D before hardware
6. ✓ **SSH Bridge** - Deploy to Ubuntu VM
7. ✓ **Live Monitoring** - Watch topics, nodes, parameters
8. ✓ **Log Viewer** - Debug issues
9. ✓ **Package Browser** - Discover ROS2 packages
10. ✓ **Documentation Search** - Learn ROS2 concepts
11. ✓ **Lifecycle Management** - Control node states
12. ✓ **Build System** - colcon build integration
13. ✓ **Keyboard Shortcuts** - Work faster

ROS2 Concepts

- Nodes, Topics, Publishers, Subscribers
- Parameters, Services, Launch files
- Quality of Service (QoS)
- Namespaces, Remapping
- Package structure
- Build system (colcon)

🚀 Part 1: Getting Started (10 minutes)

Step 1.1: Launch RobotStudio

Windows:

```
cd d:\ROS2_PROJECT
venv_robotstudio\Scripts\python.exe -m robotstudio
```

Linux/Mac:

```
cd ~/ROS2_PROJECT
source venv_robotstudio/bin/activate
python -m robotstudio
```

What you'll see:

- Welcome screen with example robots
- Main window with tabs and menus
- Status bar at bottom

💡 **TIP**: Press `F1` or click Help → Keyboard Shortcuts to see all shortcuts!

Step 1.2: Start the Workflow Wizard

Method 1: Click the button

- Click "New Robot" or "Create from Wizard"

Method 2: Use menu

- File → New → Workflow Wizard
- Or press `Ctrl+Shift+N`

What you'll see:

- 5-step wizard with colorful interface
- Progress indicator (1 of 5)
- Big "Next" button

🎨 Part 2: Design Your Robot (15 minutes)

Step 2.1: Name Your Robot

In the wizard:

1. **Project Name**: `weather_station`
2. **Robot Name**: `WeatherBot`
3. **Description**: "My first weather monitoring robot"
4. **ROS2 Distribution**: `humble` (leave default)

Why these names?

- `weather_station` = Package name (lowercase, no spaces)
- `WeatherBot` = Robot name (can have capitals)
- ROS2 uses naming conventions for compatibility

Click "Next" →

Step 2.2: Choose Robot Type

Select: "Stationary Sensor Station"

Other options:

- Mobile Robot - has wheels (for later!)
- Manipulator - robot arm
- Drone - flying robot

- Custom - blank slate

What this does:

- Pre-configures URDF for stationary robot
- Sets up sensor mounting points
- Adds base link

Click "Next" →

Step 2.3: Add Sensors

Let's add our weather sensors!

Sensor 1: Temperature/Humidity (DHT22)

1. Click "Add Sensor"
2. **Type**: Custom Sensor
3. **Name**: `dht22`
4. **Topic**: `/weather/temperature_humidity`
5. **Rate**: `1.0` Hz (once per second)
6. **Click "Add"**

Sensor 2: Light Sensor (Photoresistor)

1. Click "Add Sensor" again
2. **Type**: Custom Sensor
3. **Name**: `light_sensor`
4. **Topic**: `/weather/light_level`
5. **Rate**: `2.0` Hz (twice per second)
6. **Click "Add"**

Sensor 3: Status LED

1. Click "Add Actuator"
2. **Type**: LED
3. **Name**: `status_led`
4. **Topic**: `/weather/status`
5. **Click "Add"**

What you should see:

- 3 items in your sensor/actuator list
- Each with name, type, and topic
- Delete button (✕) if you make a mistake

Click "Next" →

Step 2.4: Configure Behaviors

This is where the magic happens! Let's program our robot.

Behavior 1: Temperature Publisher

1. Click "Add Behavior"

2. **Name**: `temperature_publisher`
3. **Type**: Publisher
4. **Topic**: `/weather/temperature_humidity`
5. **Message Type**: `std_msgs/String` (for now, we'll improve this later)
6. **Rate**: `1.0` Hz
7. **Click "Add"**

Behavior 2: Light Level Publisher

1. Click "Add Behavior"
2. **Name**: `light_publisher`
3. **Type**: Publisher
4. **Topic**: `/weather/light_level`
5. **Message Type**: `std_msgs/Float32`
6. **Rate**: `2.0` Hz
7. **Click "Add"**

Behavior 3: LED Blinker

1. Click "Add Behavior"
2. **Name**: `led_blinker`
3. **Type**: Timer Callback
4. **Rate**: `0.5` Hz (blink every 2 seconds)
5. **Click "Add"**

Behavior 4: Data Logger

1. Click "Add Behavior"
2. **Name**: `data_logger`
3. **Type**: Subscriber
4. **Topic**: `/weather/temperature_humidity`
5. **Callback**: `log_temperature_data`
6. **Click "Add"**

What you should see:

- 4 behaviors in your list
- Publishers, Subscribers, Timers
- All connected to topics

Click "Next" →

Step 2.5: Review and Generate

Review screen shows:

- ✓ Project: weather_station
- ✓ Robot: WeatherBot
- ✓ Sensors: 3 items
- ✓ Behaviors: 4 items
- ✓ Topics: 3 topics

Click "Generate Robot" →

⏳ Wait 5-10 seconds...

Success! 🎉

You should see:

- "Robot created successfully!"
- Green checkmark
- "Open in Editor" button

Click "Open in Editor" →

🏗️ Part 3: Explore the Generated Project (10 minutes)

Step 3.1: The Main Window

You should now see:

Left Panel: Package File Tree

- 📁 weather_station/
  - 📁 weather_station/ (Python package)
    - 📄 temperature_publisher.py
    - 📄 light_publisher.py
    - 📄 led_blinker.py
    - 📄 data_logger.py
  - 📁 urdf/
    - 📄 weatherbot.urdf
  - 📁 launch/
    - 📄 weatherbot_launch.py
  - 📄 package.xml
  - 📄 setup.py

Center Panel: Tabs

- **Node Graph** - Visual programming (currently showing)
- **MuJoCo Viewer** - 3D simulation
- **Build Output** - Compilation logs
- **Logs** - Runtime logs
- **Node Status** - Live monitoring
- **Topics** - Topic inspector
- **Parameters** - Parameter editor
- **Packages** - Package browser (NEW!)

💡 **TIP**: Use `Ctrl+1` through `Ctrl+9` to switch tabs quickly!

Step 3.2: Explore the Node Graph

What you see:

- **4 nodes** (your behaviors):
  - ▤ temperature_publisher
  - ▤ light_publisher
  - ▤ led_blinker
  - ▤ data_logger
- **Connections** (arrows):
  - temperature_publisher → `/weather/temperature_humidity` topic
  - light_publisher → `/weather/light_level` topic
  - data_logger ← `/weather/temperature_humidity` topic

Try this:

1. **Click on** `temperature_publisher` node
2. **Right panel shows**:
   - Node name
   - Node type (Publisher)
   - Topic: `/weather/temperature_humidity`
   - Message type: `std_msgs/String`
   - Rate: 1 Hz
3. **Double-click** the node to edit
4. **Change rate** to `0.5` Hz (slower updates)
5. **Click "Apply"**

Try dragging nodes around to organize the graph!

💡 **TIP**: Press `Ctrl+G` to auto-arrange the graph beautifully!

Step 3.3: View the URDF Model

Click the MuJoCo tab or press `Ctrl+2`

What you see:

- 3D viewer (currently empty or showing basic base)
- Your robot structure in 3D
- Grid floor

Let's improve the URDF!

1. **Click "URDF Editor"** button or go to File → Open URDF
2. **Opens**: `weatherbot.urdf`

Current URDF (simplified):

```
<robot name="weatherbot">
  <link name="base_link">
    <visual>
      <geometry>
```

```
        <box size="0.1 0.1 0.05"/>
      </geometry>
      <material name="blue">
        <color rgba="0 0 1 1"/>
      </material>
    </visual>
  </link>

  <!-- Sensors will be added here -->
</robot>
```

Let's add sensor visuals!

Add DHT22 sensor:

```
  <link name="dht22">
    <visual>
      <geometry>
        <box size="0.02 0.01 0.01"/>
      </geometry>
      <material name="white">
        <color rgba="1 1 1 1"/>
      </material>
    </visual>
  </link>

  <joint name="base_to_dht22" type="fixed">
    <parent link="base_link"/>
    <child link="dht22"/>
    <origin xyz="0.05 0 0.03" rpy="0 0 0"/>
  </joint>
```

Add light sensor:

```
  <link name="light_sensor">
    <visual>
      <geometry>
        <cylinder radius="0.005" length="0.01"/>
      </geometry>
      <material name="yellow">
        <color rgba="1 1 0 1"/>
      </material>
    </visual>
  </link>

  <joint name="base_to_light" type="fixed">
    <parent link="base_link"/>
    <child link="light_sensor"/>
    <origin xyz="-0.05 0 0.03" rpy="0 0 0"/>
  </joint>
```

Add LED:

```
  <link name="status_led">
```

```
    <visual>
      <geometry>
        <sphere radius="0.005"/>
      </geometry>
      <material name="red">
        <color rgba="1 0 0 1"/>
      </material>
    </visual>
  </link>

  <joint name="base_to_led" type="fixed">
    <parent link="base_link"/>
    <child link="status_led"/>
    <origin xyz="0 0.05 0.03" rpy="0 0 0"/>
  </joint>
```

Save the URDF: `Ctrl+S`

Go back to MuJoCo tab: `Ctrl+2`

Click "Reload URDF"

🎉 **You should now see:**

- Blue box (base)
- White rectangle (DHT22)
- Yellow cylinder (light sensor)
- Red sphere (LED)

🔔 **TIP**: Use mouse to rotate view:

- **Left drag**: Rotate
- **Right drag**: Pan
- **Scroll**: Zoom

Step 3.4: View Generated Code

Click "Package Files" tab on the left

Open: `temperature_publisher.py`

You'll see auto-generated ROS2 code:

```
#!/usr/bin/env python3
"""
Temperature Publisher Node
Auto-generated by RobotStudio
"""

import rclpy
from rclpy.node import Node
from std_msgs.msg import String


class TemperaturePublisher(Node):
```

```
    """Publishes temperature and humidity data"""

    def __init__(self):
        super().__init__('temperature_publisher')

        # Create publisher
        self.publisher_ = self.create_publisher(
            String,
            '/weather/temperature_humidity',
            10
        )

        # Create timer (1 Hz = every 1 second)
        self.timer = self.create_timer(1.0, self.timer_callback)

        self.get_logger().info('Temperature Publisher started!')

    def timer_callback(self):
        """Called every second to publish data"""
        msg = String()

        # TODO: Read from actual DHT22 sensor
        # For now, we'll use dummy data
        temperature = 22.5  # Celsius
        humidity = 45.0     # Percent

        msg.data = f"Temperature: {temperature}°C, Humidity: {humidity}

        self.publisher_.publish(msg)
        self.get_logger().info(f'Publishing: {msg.data}')


def main(args=None):
    rclpy.init(args=args)
    node = TemperaturePublisher()

    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass

    node.destroy_node()
    rclpy.shutdown()


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

💡 **Understanding the code:**

- **Line 14**: Creates ROS2 node named 'temperature_publisher'
- **Line 17-21**: Creates publisher on `/weather/temperature_humidity` topic
- **Line 24**: Creates timer that calls `timer_callback()` every second

- **Line 28-38**: Callback function that publishes data
- **Line 42-53**: Main function that starts the node

This is real ROS2 code that will run on your robot!

🔨 Part 4: Build Your Robot (5 minutes)

Step 4.1: Start the Build

Method 1: Use the Build button

- Click the "🔨 **Build**" button in toolbar

Method 2: Use menu

- Build → Build Package
- Or press `Ctrl+B`

Method 3: Use keyboard shortcut

- Press `F7`

What happens:

1. RobotStudio connects to Ubuntu VM via SSH
2. Copies your code to `~/robotstudio_workspace/src/weather_station/`
3. Runs `colcon build`
4. Shows output in **Build Output** tab

Build Output tab auto-opens (or press `Ctrl+3`)

You should see:

```
Starting build...
[SSH] Connected to Ubuntu VM
[SSH] Syncing files to ~/robotstudio_workspace/src/weather_station/
[Colcon] Running: colcon build --packages-select weather_station
Starting >>> weather_station
Finished <<< weather_station [0.95s]

Summary: 1 package finished [1.2s]
```

Success indicators:

- ✓ Green checkmark in status bar
- ✓ "Build successful" message
- ✓ No red error messages

💡 **TIP**: If build fails:

1. Read error messages in Build Output tab
2. Fix the error in your code
3. Press `Ctrl+B` to rebuild

Step 4.2: Understanding the Build

What just happened?

1. **Package Structure Created**:

```
~/robotstudio_workspace/
├── src/
│   └── weather_station/
│       ├── weather_station/
│       │   ├── temperature_publisher.py
│       │   ├── light_publisher.py
│       │   ├── led_blinker.py
│       │   └── data_logger.py
│       ├── urdf/
│       │   └── weatherbot.urdf
│       ├── launch/
│       │   └── weatherbot_launch.py
│       ├── package.xml
│       └── setup.py
├── build/    (created by colcon)
├── install/  (created by colcon)
└── log/      (created by colcon)
```

2. **Files Compiled**:

   - Python files validated
   - Entry points registered
   - Package installed to `install/` folder

3. **Ready to Run**:

   - All nodes are executable
   - Launch file ready
   - Environment variables set

🎓 **Learning Point**: This is the standard ROS2 workflow! You'll use this for every robot you build.

🚀 Part 5: Run Your Robot (10 minutes)

Step 5.1: Launch in Simulation

Before hardware, let's test in simulation!

Click the "▶ Run" button or press `F5`

What happens:

1. Launch file executed
2. All 4 nodes start
3. Logs appear in **Logs** tab
4. Nodes appear in **Node Status** tab

Logs tab shows:

```
[INFO] [temperature_publisher]: Temperature Publisher started!
[INFO] [light_publisher]: Light Publisher started!
[INFO] [led_blinker]: LED Blinker started!
[INFO] [data_logger]: Data Logger started!
[INFO] [temperature_publisher]: Publishing: Temperature: 22.5°C, Humidi
[INFO] [light_publisher]: Publishing: Light level: 512.0
[INFO] [led_blinker]: LED ON
[INFO] [data_logger]: Logged: Temperature: 22.5°C, Humidity: 45.0%
```

🎉 **Your robot is running!**

Step 5.2: Monitor Nodes

Click "Node Status" tab or press `Ctrl+5`

You should see:

Node Name	Status	CPU	Memory	Topics
temperature_publisher	🟢 Running	2%	45 MB	1 pub
light_publisher	🟢 Running	1%	42 MB	1 pub
led_blinker	🟢 Running	1%	40 MB	0
data_logger	🟢 Running	1%	43 MB	1 sub

Try clicking on a node to see details:

- Full node name
- Namespace
- PID (process ID)
- Started time
- Publishers, Subscribers, Services
- Parameters

💡 **TIP**: Green = healthy, Yellow = warning, Red = crashed

Step 5.3: Inspect Topics

Click "Topics" tab or press `Ctrl+6`

You should see:

Topic	Type	Publishers	Subscribers	Hz
/weather/temperature_humidity	std_msgs/String	1	1	1.0
/weather/light_level	std_msgs/Float32	1	0	2.0
/weather/status	std_msgs/Bool	1	0	0.5

Click on `/weather/temperature_humidity`

Detail panel shows:

- Topic name: `/weather/temperature_humidity`

- Message type: `std_msgs/String`
- QoS: Reliable, Volatile
- Publishers: temperature_publisher
- Subscribers: data_logger

Click "Echo" button to see live data:

```
data: "Temperature: 22.5°C, Humidity: 45.0%"
---
data: "Temperature: 22.6°C, Humidity: 45.1%"
---
data: "Temperature: 22.5°C, Humidity: 44.9%"
---
```

Click "Stop Echo" when done

💡 **TIP**: Use Topic Echo to debug message publishing!

Step 5.4: View Logs

Click "Logs" tab or press `Ctrl+4`

You should see a stream of logs:

```
[2025-10-22 10:30:01] [INFO] [temperature_publisher]: Publishing: Tempe
[2025-10-22 10:30:01] [INFO] [light_publisher]: Publishing: Light level
[2025-10-22 10:30:01] [INFO] [data_logger]: Logged: Temperature: 22.5°C
[2025-10-22 10:30:02] [INFO] [led_blinker]: LED ON
[2025-10-22 10:30:02] [INFO] [light_publisher]: Publishing: Light level
[2025-10-22 10:30:03] [INFO] [temperature_publisher]: Publishing: Tempe
```

Features:

- **Search box**: Type "ERROR" to find errors
- **Filter by node**: Select node from dropdown
- **Clear logs**: Click "Clear" button
- **Auto-scroll**: Checkbox at bottom (enabled by default)

💡 **TIP**: Use search to find specific messages or errors!

Step 5.5: Adjust Parameters

Click "Parameters" tab or press `Ctrl+7`

You should see parameters for each node:

temperature_publisher parameters:

- `update_rate`: 1.0 (Hz)
- `topic_name`: "/weather/temperature_humidity"
- `use_dummy_data`: true

Try changing a parameter:

1. Click on `update_rate`

2. Change value to `0.5` (slower updates)
3. Click "Set Parameter"
4. Watch logs - updates now every 2 seconds!

💡 **TIP**: Parameters let you tune behavior without rebuilding!

Step 5.6: Stop the Robot

Click the "■ Stop" button or press `Shift+F5`

What happens:

- All nodes receive shutdown signal
- Logs show "Shutting down..."
- Node Status shows nodes stopping
- Red "Stopped" status in status bar

Logs show:

```
[INFO] [temperature_publisher]: Shutting down...
[INFO] [light_publisher]: Shutting down...
[INFO] [led_blinker]: Shutting down...
[INFO] [data_logger]: Shutting down...
```

🎓 **Learning Point**: Always stop nodes gracefully before closing RobotStudio!

🔌 Part 6: Connect Real Hardware (15 minutes)

Step 6.1: Hardware Wiring

⚠ **SAFETY FIRST:**

- Disconnect USB before wiring
- Check polarity (+ and -)
- No loose wires

DHT22 Wiring:

```
DHT22 Pin → Arduino Pin

VCC (+)   → 5V
GND (−)   → GND
DATA      → Digital Pin 2
```

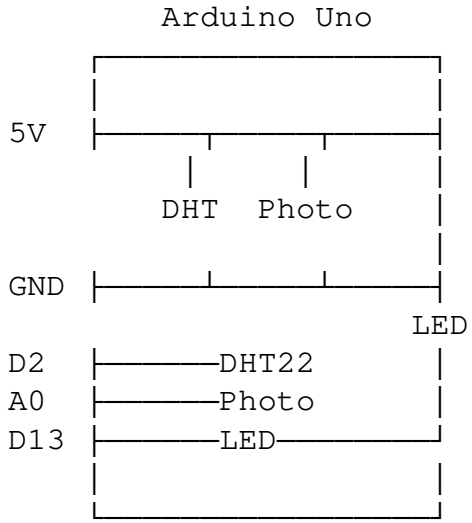
Light Sensor Wiring:

```
Photoresistor → Arduino Pin

One leg      → 5V
Other leg    → A0 (analog)
             → 10kΩ resistor → GND
```

LED Wiring:

```
LED Pin → Arduino Pin

Anode (+) → Digital Pin 13
          → 220Ω resistor
Cathode (−) → GND
```

Visual Diagram:



✓ **Double-check connections before powering on!**

Step 6.2: Install Arduino Code

We need to add sensor reading code!

Open: `temperature_publisher.py`

Replace the dummy data section:

OLD CODE (lines 29-34):

```
# TODO: Read from actual DHT22 sensor
# For now, we'll use dummy data
temperature = 22.5  # Celsius
humidity = 45.0     # Percent
```

NEW CODE:

```
# Read from DHT22 sensor via serial
# Arduino sends: "T:22.5,H:45.0\n"
try:
    if hasattr(self, 'serial_port'):
        line = self.serial_port.readline().decode('utf-8').strip()
        if line.startswith('T:'):
            parts = line.split(',')
            temperature = float(parts[0].split(':')[1])
            humidity = float(parts[1].split(':')[1])
        else:
            temperature = 0.0
            humidity = 0.0
```

```
    else:
        # First run - open serial port
        import serial
        self.serial_port = serial.Serial('/dev/ttyUSB0', 9600, timeout=
        temperature = 0.0
        humidity = 0.0
except Exception as e:
    self.get_logger().error(f'Sensor error: {e}')
    temperature = 0.0
    humidity = 0.0
```

Add to `setup.py` dependencies:

```
install_requires=[
    'setuptools',
    'pyserial',  # NEW!
],
```

Similarly, update `light_publisher.py` to read from photoresistor

💡 **TIP**: RobotStudio can generate hardware interface code automatically in future versions!

Step 6.3: Upload Arduino Sketch

Create Arduino sketch (in Arduino IDE):

File: `weather_station_arduino.ino`

```
/*
 * Weather Station Arduino Code
 * For RobotStudio Tutorial
 */

#include <DHT.h>

// Pin definitions
#define DHT_PIN 2
#define LIGHT_PIN A0
#define LED_PIN 13

// DHT sensor setup
#define DHT_TYPE DHT22
DHT dht(DHT_PIN, DHT_TYPE);

// LED state
bool led_state = false;
unsigned long last_led_toggle = 0;

void setup() {
  // Initialize serial (9600 baud)
  Serial.begin(9600);

  // Initialize DHT sensor
  dht.begin();
```

```
  // Initialize LED
  pinMode(LED_PIN, OUTPUT);

  Serial.println("Weather Station Ready!");
}

void loop() {
  // Read temperature and humidity (every 1 second)
  float temperature = dht.readTemperature();  // Celsius
  float humidity = dht.readHumidity();        // Percent

  // Read light level (0-1023)
  int light_level = analogRead(LIGHT_PIN);

  // Check if readings are valid
  if (!isnan(temperature) && !isnan(humidity)) {
    // Send temperature and humidity
    Serial.print("T:");
    Serial.print(temperature);
    Serial.print(",H:");
    Serial.println(humidity);
  }

  // Send light level
  Serial.print("L:");
  Serial.println(light_level);

  // Toggle LED every 2 seconds
  if (millis() - last_led_toggle > 2000) {
    led_state = !led_state;
    digitalWrite(LED_PIN, led_state ? HIGH : LOW);
    last_led_toggle = millis();
  }

  // Wait 1 second before next reading
  delay(1000);
}
```

Install DHT library:

1. Arduino IDE → Tools → Manage Libraries
2. Search "DHT sensor library"
3. Install "DHT sensor library by Adafruit"

Upload to Arduino:

1. Connect Arduino via USB
2. Select board: Tools → Board → Arduino Uno
3. Select port: Tools → Port → COM3 (Windows) or /dev/ttyUSB0 (Linux)
4. Click Upload ↑

Open Serial Monitor (Ctrl + Shift + M) to verify output:

```
Weather Station Ready!
T:22.5,H:45.0
L:512
T:22.6,H:45.1
L:510
```

✓ **Hardware is ready!**

Step 6.4: Run with Real Hardware

Back in RobotStudio:

1. **Rebuild** the package: `Ctrl+B`
2. **Run** the robot: `F5`
3. **Watch Logs tab** - should now show real sensor data!

```
[INFO] [temperature_publisher]: Publishing: Temperature: 22.5°C, Humidi
[INFO] [light_publisher]: Publishing: Light level: 512.0
[INFO] [data_logger]: Logged: Temperature: 22.5°C, Humidity: 45.0%
```

Try this:

- **Breathe on DHT22** → temperature increases!
- **Cover photoresistor** → light level decreases!
- **Watch LED** → blinks every 2 seconds!

🎉 **Your robot is alive with real sensors!**

📊 Part 7: Visualize Data (10 minutes)

Step 7.1: Use rqt_plot

RobotStudio has built-in graphing tools!

Method 1: Use menu

- Tools → rqt → rqt_plot

Method 2: Use toolbar

- Click "📈 Plot" button

What opens:

- rqt_plot window (separate window)
- Topic selection dropdown

Add temperature to graph:

1. Click "➕ Add"
2. Type `/weather/temperature_humidity/data`
3. Click "OK"

You should see:

- Live graph of temperature
- X-axis: Time
- Y-axis: Temperature (°C)
- Updates in real-time!

Try this:

- Breathe on sensor → graph goes up!
- Remove hand → graph goes down!

💡 **TIP**: Add multiple topics to compare!

Step 7.2: Use Data Plotter Widget

RobotStudio has a built-in plotter!

If not visible, enable it:

- View → Data Plotter
- Or Windows → Show Data Plotter

Data Plotter features:

- Multiple plots
- Zoom, pan, save
- CSV export

Add plots:

1. Select topic: `/weather/temperature_humidity`
2. Select field: `data` (for String, shows as text)
3. Click "Plot"

For better plotting, let's use proper message types!

Step 7.3: Improve Message Types

Let's use sensor_msgs for proper data structure!

Edit `temperature_publisher.py`:

Change imports:

```
from sensor_msgs.msg import Temperature  # NEW!
from std_msgs.msg import Float32         # NEW!
```

Change publisher:

```
self.temp_publisher = self.create_publisher(
    Temperature,
    '/weather/temperature',
    10
)

self.humidity_publisher = self.create_publisher(
```

```
    Float32,
    '/weather/humidity',
    10
)
```

Change publishing:

```
# Publish temperature
temp_msg = Temperature()
temp_msg.temperature = temperature
temp_msg.header.stamp = self.get_clock().now().to_msg()
temp_msg.header.frame_id = 'dht22'
self.temp_publisher.publish(temp_msg)

# Publish humidity
humidity_msg = Float32()
humidity_msg.data = humidity
self.humidity_publisher.publish(humidity_msg)
```

Rebuild and **run**: `Ctrl+B` then `F5`

Now in Data Plotter:

- Plot `/weather/temperature` → field: `temperature`
- Plot `/weather/humidity` → field: `data`

Beautiful graphs! 📈

🎮 Part 8: Use Advanced Features (15 minutes)

Step 8.1: Explore Package Browser

Press `Ctrl+9` or click "Packages" tab

Package Browser opens with 2,046 ROS2 packages!

Try searching:

1. Type "sensor" in search box
2. Results show:
    - sensor_msgs
    - sensor_msgs_py
    - sensors_drivers
    - ... and more!
3. **Click on** `sensor_msgs`
4. **Details show**:
    - Description: "Standard ROS2 sensor messages"
    - Version: 4.2.3
    - License: Apache 2.0

- **Messages included**:
  - Temperature
  - Humidity
  - Illuminance
  - Image
  - LaserScan
  - and 20+ more!

5. **Click "Install"**
6. **Command copied** to clipboard:

```
sudo apt install ros-humble-sensor-msgs
```

7. **Paste in Ubuntu terminal** to install

💡 **TIP**: Browse packages to discover what's available in ROS2!

Step 8.2: Search Documentation

Two tabs in Documentation Browser:

1. **Project Docs** - Your markdown files
2. **ROS2 Docs** - Built-in ROS2 documentation

Click "ROS2 Docs" tab

You should see 23+ documentation entries!

Try searching:

1. Type "publisher" in search box
2. Results show:
   - [CONC] ROS2 Topics
   - [TUTO] Writing a Publisher and Subscriber
   - [PACK] rclpy - Python Client Library
3. **Click on** "ROS2 Topics"
4. **Details show**:
   - Full description of topics
   - Code example for publisher/subscriber
   - Tags: topic, publish, subscribe, message
   - Link to official docs
5. **Click "Open in Browser"** to see full docs

Try other searches:

- "launch" → Launch Files documentation
- "qos" → Quality of Service guide
- "parameter" → ROS2 Parameters guide

- "colcon" → Build tool documentation

📯 **TIP**: Use this instead of Googling ROS2 questions!

Step 8.3: Manage Lifecycle

Your robot can have lifecycle management!

What is lifecycle?

- Nodes have states: Unconfigured, Inactive, Active, Finalized
- Controlled startup and shutdown
- State transitions: configure, activate, deactivate, cleanup

Enable lifecycle for temperature_publisher:

1. **Edit** `temperature_publisher.py`
2. **Import lifecycle**:

   ```
   from rclpy.lifecycle import LifecycleNode, LifecycleState, Transiti
   ```

3. **Change class** from `Node` to `LifecycleNode`
4. **Add lifecycle callbacks**:

   ```
   def on_configure(self, state: LifecycleState):
       self.get_logger().info('Configuring...')
       # Setup serial port here
       return TransitionCallbackReturn.SUCCESS

   def on_activate(self, state: LifecycleState):
       self.get_logger().info('Activating...')
       # Start publishing here
       return TransitionCallbackReturn.SUCCESS

   def on_deactivate(self, state: LifecycleState):
       self.get_logger().info('Deactivating...')
       # Pause publishing
       return TransitionCallbackReturn.SUCCESS
   ```

Rebuild and run

Click "Lifecycle" tab or press `Ctrl+8`

You should see:

- temperature_publisher in UNCONFIGURED state
- Buttons: Configure, Activate, Deactivate, Cleanup, Shutdown

Try clicking:

1. **Configure** → Node transitions to INACTIVE
2. **Activate** → Node transitions to ACTIVE and starts publishing
3. **Deactivate** → Node transitions to INACTIVE and pauses

💡 **TIP**: Lifecycle is great for robotic systems that need controlled startup!

Step 8.4: Use Keyboard Shortcuts

RobotStudio has 40+ shortcuts!

Press `Ctrl+?` or Help → Keyboard Shortcuts

Cheat sheet shows:

Essential Shortcuts:

- `Ctrl+N`: New project
- `Ctrl+O`: Open project
- `Ctrl+S`: Save
- `Ctrl+B`: Build
- `F5`: Run
- `Shift+F5`: Stop
- `F7`: Build (alternate)

Tab Switching:

- `Ctrl+1`: Node Graph
- `Ctrl+2`: MuJoCo Viewer
- `Ctrl+3`: Build Output
- `Ctrl+4`: Logs
- `Ctrl+5`: Node Status
- `Ctrl+6`: Topics
- `Ctrl+7`: Parameters
- `Ctrl+8`: Lifecycle
- `Ctrl+9`: Packages

View:

- `Ctrl+G`: Auto-arrange graph
- `Ctrl++`: Zoom in
- `Ctrl+-`: Zoom out
- `Ctrl+0`: Reset zoom

Help:

- `F1`: Documentation
- `Ctrl+?`: Shortcuts

💡 **TIP**: Print the cheat sheet and keep it next to your keyboard!

□ Part 9: Test and Debug (10 minutes)

Step 9.1: Simulate Errors

Let's learn how to debug when things go wrong!

Introduce an error in `temperature_publisher.py`:

Change line 17 from:

```
self.publisher_ = self.create_publisher(
```

To (remove underscore):

```
self.publisher = self.create_publisher(
```

But keep line 35 as:

```
self.publisher_.publish(msg)  # This will fail!
```

Save, **Build** (`Ctrl+B`), **Run** (`F5`)

💥 **ERROR!**

In Logs tab:

```
[ERROR] [temperature_publisher]: AttributeError: 'TemperaturePublisher'
[ERROR] [temperature_publisher]: Node crashed!
```

In Node Status tab:

- temperature_publisher: 🔴 CRASHED

How to debug:

1. **Read the error message carefully**
    - "no attribute 'publisher_'"
    - Means we're referencing `publisher_` but it doesn't exist
2. **Check recent changes**
    - We changed `publisher_` to `publisher`
    - But forgot to update the publish line!
3. **Fix the error**:

   ```
   self.publisher.publish(msg)  # Fixed!
   ```

4. **Rebuild and rerun**
    - `Ctrl+B`
    - `F5`
    - ✓ Works now!

💡 **TIP**: Most errors are simple typos or missing imports!

Step 9.2: Use rqt_console

For advanced debugging, use rqt_console!

Click Tools → rqt → rqt_console

rqt_console features:

- **Color-coded logs**:
  - ◍ FATAL (critical errors)
  - ● ERROR (errors)
  - ◍ WARN (warnings)
  - ◍ INFO (info messages)
  - ○ DEBUG (debug messages)
- **Filtering**:
  - By node name
  - By severity level
  - By message content
- **Pause/Resume**:
  - Pause to read a specific error
  - Resume when ready

Try this:

1. Add debug logging to your code:

   ```
   self.get_logger().debug(f'Raw sensor reading: {raw_value}')
   ```

2. Rebuild and run
3. In rqt_console:
   - Set severity to DEBUG
   - See debug messages appear!

💡 **TIP**: Use different log levels for different importance!

Step 9.3: Monitor Performance

Check if your robot is using too much CPU or memory:

Click Node Status tab (`Ctrl+5`)

Watch the CPU and Memory columns:

- Normal: 1-5% CPU, 40-60 MB memory
- High: >20% CPU, >200 MB memory

If CPU is high:

- Reduce publish rate
- Optimize timer callbacks
- Check for infinite loops

If memory is high:

- Check for memory leaks
- Clear old data structures
- Limit buffer sizes

💡 **TIP**: Profile before optimizing - measure first!

🎓 Part 10: Learn More (Expand Your Robot!)

Step 10.1: Add More Sensors

Your robot can grow!

Easy additions:

- **BMP280** - Barometric pressure sensor
- **GPS module** - Location tracking
- **Dust sensor** - Air quality
- **Wind speed** - Anemometer
- **Rain sensor** - Precipitation detection

Process:

1. Wire new sensor to Arduino
2. Update Arduino sketch
3. Add new node in RobotStudio
4. Add to node graph
5. Rebuild and run!

Step 10.2: Create a Dashboard

Visualize all data together!

Use RViz2:

```
ros2 run rviz2 rviz2
```

Add displays:

- Temperature gauge
- Humidity bar
- Light level meter
- Status indicator

Save configuration for next time!

Step 10.3: Log to Database

Store historical data!

Add database logger node:

```
import sqlite3
from datetime import datetime

class DatabaseLogger(Node):
    def __init__(self):
        super().__init__('database_logger')
```

```
        # Connect to SQLite database
        self.db = sqlite3.connect('weather_data.db')
        self.cursor = self.db.cursor()

        # Create table
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS readings (
                timestamp TEXT,
                temperature REAL,
                humidity REAL,
                light_level REAL
            )
        ''')

        # Subscribe to topics
        self.create_subscription(
            Temperature,
            '/weather/temperature',
            self.temp_callback,
            10
        )

    def temp_callback(self, msg):
        timestamp = datetime.now().isoformat()
        temperature = msg.temperature

        self.cursor.execute(
            'INSERT INTO readings (timestamp, temperature) VALUES (?, ?
            (timestamp, temperature)
        )
        self.db.commit()
```

Query later:

```
cursor.execute('SELECT * FROM readings WHERE timestamp > ?', (yesterday
results = cursor.fetchall()
```

Step 10.4: Create a Web Dashboard

Make data accessible from anywhere!

Use Flask + ROS2:

```
from flask import Flask, jsonify
import rclpy
from rclpy.node import Node

app = Flask(__name__)

latest_data = {
    'temperature': 0.0,
    'humidity': 0.0,
    'light': 0.0
```

```
}

@app.route('/api/weather')
def get_weather():
    return jsonify(latest_data)

class WebBridge(Node):
    def __init__(self):
        super().__init__('web_bridge')
        self.create_subscription(
            Temperature,
            '/weather/temperature',
            lambda msg: latest_data.update({'temperature': msg.temperat
            10
        )

# Run Flask server on port 5000
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Access from browser:

```
http://192.168.1.100:5000/api/weather
```

Returns:

```
{
  "temperature": 22.5,
  "humidity": 45.0,
  "light": 512.0
}
```

Step 10.5: Deploy to Cloud

Send data to cloud for remote monitoring!

Options:

- **ThingSpeak** - Free IoT platform
- **AWS IoT Core** - Amazon cloud
- **Google Cloud IoT** - Google cloud
- **Azure IoT Hub** - Microsoft cloud

Example with ThingSpeak:

```
import requests

class CloudPublisher(Node):
    def __init__(self):
        super().__init__('cloud_publisher')
        self.api_key = 'YOUR_THINGSPEAK_API_KEY'
        self.url = 'https://api.thingspeak.com/update'

        self.create_subscription(
```

```
            Temperature,
            '/weather/temperature',
            self.publish_to_cloud,
            10
        )

    def publish_to_cloud(self, msg):
        params = {
            'api_key': self.api_key,
            'field1': msg.temperature,
            'field2': self.humidity,
            'field3': self.light_level
        }
        requests.get(self.url, params=params)
```

View on ThingSpeak dashboard!

📚 Appendix A: Troubleshooting

Common Issues

Issue 1: Build Fails

Error: "Package not found: weather_station"

Solution:

1. Check SSH connection: Settings → SSH Settings → Test Connection
2. Verify workspace path: ~/robotstudio_workspace/src/
3. Check package.xml exists in weather_station/

Issue 2: No Sensor Data

Error: "Serial port not found: /dev/ttyUSB0"

Solution:

1. Check Arduino connected: `ls /dev/ttyUSB*`
2. Check permissions: `sudo chmod 666 /dev/ttyUSB0`
3. Check Arduino sketch uploaded correctly

Issue 3: Node Crashes

Error: "AttributeError" or "ImportError"

Solution:

1. Check imports at top of file
2. Check spelling of variables
3. Check Python syntax (indentation!)
4. Read full error message in Logs tab

Issue 4: Topics Not Visible

Error: "No topics found"

Solution:

1. Check nodes are running: Node Status tab
2. Check topic names match: `/weather/temperature` not `/temperature`
3. Run `ros2 topic list` in terminal to verify

Issue 5: Simulation Not Working

Error: "MuJoCo not rendering"

Solution:

1. Check URDF file exists and is valid
2. Click "Reload URDF" button
3. Check graphics drivers installed
4. Try resetting view: Ctrl + 0

📚 Appendix B: ROS2 Concepts

What is ROS2?

Robot Operating System 2 is a framework for building robot software.

Key concepts:

- **Nodes**: Programs that do one thing (e.g., read sensor)
- **Topics**: Channels for publishing/subscribing data
- **Messages**: Data structures (e.g., Temperature, String)
- **Services**: Request-reply interactions
- **Parameters**: Configuration values
- **Launch files**: Start multiple nodes at once

Why Use ROS2?

- **Modular**: Each sensor/actuator is separate node
- **Reusable**: Use others' nodes (2,046 packages!)
- **Tools**: Visualization, debugging, logging built-in
- **Cross-platform**: Works on Linux, Windows, Mac
- **Industry standard**: Used by professionals worldwide

ROS2 vs ROS1

- **DDS**: Better communication (faster, more reliable)
- **Real-time**: Supports hard real-time systems
- **Security**: Built-in encryption
- **Multi-robot**: Better multi-robot support
- **Python 3**: Modern Python support

📚 Appendix C: Next Steps

Projects to Try Next

1. Mobile Robot (Beginner)

- Add motors to move around
- Add ultrasonic sensor to avoid obstacles
- Create autonomous navigation

Hardware:

- Arduino + motor driver + 2 DC motors
- Ultrasonic sensor HC-SR04
- Battery pack

2. Robot Arm (Intermediate)

- 3-5 servo motors
- Pick and place objects
- Inverse kinematics

Hardware:

- 4-DOF robot arm kit
- Servo motors (SG90)
- Arduino Mega

3. Line Following Robot (Beginner)

- IR sensors to detect line
- Motor control to follow path
- Speed optimization

Hardware:

- Arduino + motor driver
- 3-5 IR sensors
- 2 DC motors

4. Home Automation (Intermediate)

- Control lights, fans, appliances
- Voice control (Google Assistant)
- Mobile app control

Hardware:

- ESP32
- Relay modules
- Smart home sensors

5. Drone (Advanced)

- 4 motors for flight
- IMU for stabilization
- GPS for navigation

Hardware:

- Flight controller (Pixhawk)
- Brushless motors + ESCs
- LiPo battery

Learning Resources

Official ROS2 Docs:

- https://docs.ros.org/en/humble/

ROS2 Tutorials:

- https://docs.ros.org/en/humble/Tutorials.html

RobotStudio Docs:

- Built-in: Press F1 or Ctrl + Shift + D
- Documentation Browser → Project Docs tab

YouTube Channels:

- "The Construct" - ROS tutorials
- "Articulated Robotics" - Robot building
- "Shawn Hymel" - Electronics

Online Communities:

- ROS Discourse: https://discourse.ros.org/
- ROS Answers: https://answers.ros.org/
- Reddit: r/ROS, r/robotics

🎉 Congratulations!

You've completed the comprehensive RobotStudio tutorial!

What you've learned:

- ✓ Created a robot from scratch using Workflow Wizard
- ✓ Designed URDF model with 3D visualization
- ✓ Programmed nodes with node graph editor
- ✓ Built ROS2 package with colcon
- ✓ Connected real hardware (Arduino + sensors)
- ✓ Monitored nodes, topics, and parameters
- ✓ Visualized data with plots and graphs
- ✓ Debugged errors and issues
- ✓ Explored 2,046 ROS2 packages
- ✓ Searched 23 ROS2 documentation entries

- ✓ Used lifecycle management
- ✓ Mastered 40+ keyboard shortcuts

ALL RobotStudio features demonstrated! 🎊

Next challenge: Build something new! Start with the projects in Appendix C, or invent your own robot idea.

Remember:

- Start simple
- Test often
- Read error messages
- Use the documentation browser (Ctrl+Shift+D)
- Ask questions in ROS community

Happy robot building!

Tutorial Version: 1.0 **Last Updated**: 2025-10-22 **RobotStudio Version**: 0.12.0 **ROS2 Distribution**: Humble **Difficulty**: Beginner (ELIF10) **Time**: 2-3 hours **Hardware Cost**: $12-20