

ENPM605: PYTHON APPLICATIONS FOR ROBOTICS

RWA #4

v1.0

Lecturer: Z. Kootbally

School: University of Maryland

Semester/Year: Spring/2025



MARYLAND APPLIED
GRADUATE ENGINEERING

Table of Contents

- ⦿ Guidelines

- ⦿ Objectives

- ⦿ Packages

- ⦿ Deliverable

- ⦿ Exercise #1

 - ⦿ Data Publisher

 - ⦿ Data Subscriber

 - ⦿ Moving Average Subscriber

 - ⦿ Message Rate Subscriber

 - ⦿ Counter Monitor Subscriber

- ⦿ Exercise #2

 - ⦿ Temperature Publisher

 - ⦿ Humidity Publisher

 - ⦿ Alert Generator

 - ⦿ Lifecycle Manager















- ⦿ Grading Rubric

 - ⦿ Deductions

✿ Changelog

- ⦿ **v1.0**: Original version.

✿ Conventions

bad practice	
best practice	
code syntax	
example	
exercise	
file	
folder	
guideline	
note	
question	
task	
terminology	
warning	
web link	<u>link</u>
package	

ROS node	n /node
ROS topic	t /topic
ROS message	m /message
ROS parameter	p /parameter
ROS frame	f /frame

✱ Guidelines

This assignment must be completed as a *group*. All instructions outlined below must be followed precisely. Failure to adhere to these rules may result in a grade of *zero*.






- ⦿ You may not reuse solutions or components created by others for similar assignments.
- ⦿ Your work must remain confidential — do not share your files or code with anyone.
- ⦿ If you choose to use GitHub, your repository must be set to *private*.
- ⦿ You are encouraged to discuss general concepts and ideas with peers, but sharing specific implementation details is strictly prohibited.
- ⦿ *The use of AI-generated code (e.g., ChatGPT, Copilot) is not allowed.*

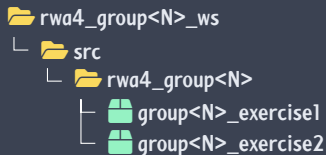
✧ Objectives

These ROS2 exercises develop skills in robotics middleware with focus on message passing, concurrency, and performance optimization.

- ⦿ **Exercise 1** establishes proficiency in the publisher-subscriber pattern within multi-threaded environments, teaching parallel data processing while maintaining system reliability.
- ⦿ **Exercise 2** advances these concepts by challenging students to implement priority-based scheduling through custom executors, addressing real-time constraints in complex systems.

✱ Packages

- ⦿ Create a new workspace, for instance  rwa4_group<N>_ws.
 - Create a regular folder in  rwa4_group<N>_ws/src:  rwa4_group<N>
 - ▷  group<N>_exercise1
 - ▷  group<N>_exercise2



- ⦿ Change <N> to your actual group number (e.g., 1, 2).
- ⦿ Ensure you have the exact structure as shown in the file tree above.

✖ Deliverable

Compress  rwa4_group<N> into *a single file* and submit it on Canvas by the due date.


Late submissions will incur a penalty according to the guidelines specified in the syllabus, with exceptions for any valid reason.

- ⦿ Valid reasons include a doctor's note, proof of travel, or a note from a professor/MAGE.


Students with special circumstances may submit their assignments late without incurring any penalties. However, it is required that these students inform me in advance of their intention to submit their work past the deadline.

Exercise #1

✿ Exercise #1

- ⦿ **Objective:** Create a system with one publisher and three subscribers running concurrently using ROS2's multi-threaded executor. The system will publish standard ROS2 messages and implement different processing strategies in each subscriber.
- ⦿ **Package:**  group<N>_exercise1

✿ Data Publisher

- ◉ **Script:**  `data_publisher.py`
- ◉ **Objective:** Create a node that publishes three types of standard ROS2 messages. Each publisher publishes data at a frequency of 20Hz.
 - **publisher #1:** publishes messages `m std_msgs/msg/Header` on `t data/timestamp`
 - ▷ Publish the current time and the frame id `data_frame`.
 - **publisher #2:** publishes a random `m std_msgs/msg/Float64` (between 0-100) on `t data/random`
 - **publisher #3:** publishes an incremental counter of type `m std_msgs/msg/Int64` on `t data/counter`. Once in a while, skip a value. One way to achieve this is to generate a random number between 0-1. If this random number is less than 0.1 (which has a 10% probability of happening):
 - ▷ The counter is incremented by 2 instead of 1, effectively skipping a value.
 - ▷ The system logs a message indicating it intentionally skipped a value.
 - ▷ Example:

```
[INFO] [1713811245.678910] [data_publisher]: Data Publisher Node has been initialized
[INFO] [1713811248.321098] [data_publisher]: Intentionally skipped a counter value to 42
```

✱ Moving Average Subscriber

◎ *Script:*  `moving_average_subscriber.py`

◎ *Objective:*

- Create a node with one subscriber to `t data/random`
- Use a sliding window of 10 values.
- Log the current window average when the window is full for the first time.
- Subsequent logs only occur when significant changes in the current average are detected.
 - ▷ *Significant* is defined as a change greater or lower than 5% from the last reported average.


A *sliding window of 10 values* refers to a data processing technique where only the 10 most recent values are considered for calculations at any given time.

- ◎ The system maintains a fixed-size collection (window) that holds exactly 10 values.
- ◎ When a new value arrives, it is added to the window.
- ◎ If the window already contains 10 values, the oldest value is automatically removed.
- ◎ Calculations (such as computing an average) are performed using only the values currently in the window.

Demonstration

```
[INFO] [1713811245.876543] [moving_average_subscriber]: Moving Average Subscriber initialized with window size 10
[INFO] [1713811247.123456] [moving_average_subscriber]: Window full for first time. Current moving average: 52.37
[INFO] [1713811259.654321] [moving_average_subscriber]: Significant change detected (+7.8%). Current moving average: 56.45
[INFO] [1713811268.987654] [moving_average_subscriber]: Significant change detected (-9.2%). Current moving average: 51.26
[INFO] [1713811287.345678] [moving_average_subscriber]: Significant change detected (+11.4%). Current moving average: 57.10
```

✱ Message Rate Subscriber

◎ *Script:*  message_rate_subscriber.py

◎ *Objective:*

- Create a node with one subscriber to `t data/timestamp`
- Calculate and report the publication rate every second. Perform this in a timer callback running at 1Hz.
- Log a warning for deviations from the expected 20Hz rate.
 - ▷ Based on the implementation, the system flags a deviation when the measured rate differs from the expected 20Hz by a 10% deviation. So the acceptable range would be between 18Hz and 22Hz, and anything outside this range would trigger a warning log.

When calculating message publication rates, using the timestamp from the message header provides a more accurate measurement than using the time when messages are received. This approach:


- ◎ Reflects the actual publishing rate at the source.
- ◎ Eliminates distortions caused by network delays or processing bottlenecks.
- ◎ Provides a more consistent measurement that is less affected by the subscriber's performance.



Demonstration

```
[INFO] [1713811245.987654] [message_rate_subscriber]: Message Rate Subscriber has been initialized
[INFO] [1713811247.012345] [message_rate_subscriber]: Current message rate: 19.86 Hz (based on last
20 messages)
[INFO] [1713811248.233221] [message_rate_subscriber]: Current message rate: 19.31 Hz (based on last
20 messages)
[INFO] [1713811249.143221] [message_rate_subscriber]: Current message rate: 18.63 Hz (based on last
20 messages)
[WARN] [1713811250.098765] [message_rate_subscriber]: Message rate deviation detected: 17.25 Hz (ex-
pected ~20Hz)
[INFO] [1713811251.023837] [message_rate_subscriber]: Current message rate: 21.29 Hz (based on last
20 messages)
```

✱ Counter Monitor Subscriber


- ◉ *Script:*  counter_monitor_subscriber.py
- ◉ *Objective:*
 - Create a node with one subscriber to `t data/counter`
 - Track counter values for discontinuities.
 - Log warnings when counter values are skipped.
 - Report the first received counter value.

⚙️ Demonstration

```
[INFO] [1713811246.765432] [counter_monitor_subscriber]: Counter Monitor Subscriber has been initialized
[INFO] [1713811246.876543] [counter_monitor_subscriber]: First counter value received: 1
[WARN] [1713811248.432109] [counter_monitor_subscriber]: Counter discontinuity detected! Expected: 41, Received: 42, Skipped: 1 value(s)
```


Exercise #2

✿ Exercise #2

- ◎ **Objective:** Create a ROS2 system demonstrating proper lifecycle management in a publisher-subscriber architecture. Students will implement a network of nodes that publish and monitor environmental sensor data (temperature and humidity), with a focus on proper resource management throughout the nodes' lifecycles.
- ◎ **Package:**  group<N>_exercise2

✱ Temperature Publisher

⦿ *Script:*  temperature_publisher.py

⦿ *Objective:*

- Create a lifecycle-managed node that publishes to `t sensors/temperature`.
- Implement complete lifecycle management (configure, activate, deactivate, cleanup).
- Only publish temperature data when the node is in active state.
 - ▷ Publisher created during configure.
 - ▷ Timer for publishing (1Hz) created during activate.
 - ▷ Timer destroyed during deactivate to stop publishing.
 - ▷ Publisher resources released during cleanup.

⦿ *Message Details:*

- Type: `m sensor_msgs/msg/Temperature`
- Temperature range: $22.0^{\circ}\text{C} \pm 3.0^{\circ}\text{C}$ (random value between $19.0 - 25.0^{\circ}\text{C}$).
- Variance: 0.1 (fixed value).
- Frame ID: *temperature_sensor*.
- Header timestamp: Current time when message is created.



Demonstration

```
[INFO] [1713811245.123456] [temperature_publisher]: Temperature publisher initialized
[INFO] [1713811245.234567] [temperature_publisher]: Configuring temperature publisher
[INFO] [1713811245.345678] [temperature_publisher]: Temperature publisher configured successfully
[INFO] [1713811245.456789] [temperature_publisher]: Activating temperature publisher
[INFO] [1713811245.567890] [temperature_publisher]: Temperature publisher activated successfully
[INFO] [1713811246.678901] [temperature_publisher]: Publishing temperature: 21.4°C
[INFO] [1713811247.789012] [temperature_publisher]: Publishing temperature: 23.8°C
[INFO] [1713811248.890123] [temperature_publisher]: Publishing temperature: 20.2°C
```

✱ Humidity Publisher

◎ *Script:*  humidity_publisher.py

◎ *Objective:*

- Create a lifecycle-managed node that publishes to `t sensors/humidity`.
- Implement complete lifecycle management (configure, activate, deactivate, cleanup).
- Only publish humidity data when the node is in active state.
 - ▷ Publisher created during configure.
 - ▷ Timer for publishing (1Hz) created during activate.
 - ▷ Timer destroyed during deactivate to stop publishing.
 - ▷ Publisher resources released during cleanup.

◎ *Message Details:*

- Type: `m sensor_msgs/msg/RelativeHumidity`
- Humidity range: 45.0% ± 10.0% (random value between 35.0-55.0%).
- Variance: 2.0 (fixed value).
- Frame ID: *humidity_sensor*.
- Header timestamp: Current time when message is created.



Demonstration

```
[INFO] [1713811245.234567] [humidity_publisher]: Humidity publisher initialized
[INFO] [1713811245.345678] [humidity_publisher]: Configuring humidity publisher
[INFO] [1713811245.456789] [humidity_publisher]: Humidity publisher configured successfully
[INFO] [1713811245.567890] [humidity_publisher]: Activating humidity publisher
[INFO] [1713811245.678901] [humidity_publisher]: Humidity publisher activated successfully
[INFO] [1713811246.789012] [humidity_publisher]: Publishing humidity: 44.7%
[INFO] [1713811247.890123] [humidity_publisher]: Publishing humidity: 38.2%
[INFO] [1713811248.901234] [humidity_publisher]: Publishing humidity: 47.5%
```

✱ Alert Generator

◎ *Script:*  `alert_generator.py`

◎ *Objective:*

- Create a lifecycle-managed node with subscribers to `t sensors/temperature` and `t sensors/humidity`.
- Monitor sensor values and generate alerts when thresholds are exceeded.
- Report when values return to normal range.
 - ▷ Temperature thresholds: 18°C (low) and 25°C (high).
 - ▷ Humidity thresholds: 30% (low) and 65% (high).
- Only process data when node is in active state.

The Alert Generator demonstrates two key concepts:

- ◎ Only processing data in the active state prevents erroneous alerts during system startup/shutdown.
- ◎ Maintaining alert status (triggered/normal) avoids duplicate alerts and allows proper reporting when conditions return to normal.

Demonstration

```
[INFO] [1713811245.987654] [alert_generator]: Alert generator initialized
[INFO] [1713811246.234567] [alert_generator]: Configuring alert generator
[INFO] [1713811246.345678] [alert_generator]: Alert generator configured successfully
[INFO] [1713811246.456789] [alert_generator]: Activating alert generator
[INFO] [1713811246.567890] [alert_generator]: Alert generator activated successfully
[INFO] [1713811250.678901] [alert_generator]: All environmental conditions within normal ranges
[WARN] [1713811252.890123] [alert_generator]: ALERT: Temperature exceeding threshold: 25.7°C
[INFO] [1713811260.901234] [alert_generator]: Temperature returned to normal range: 24.2°C
[WARN] [1713811270.012345] [alert_generator]: ALERT: Humidity below threshold: 28.3%
```


✱ Lifecycle Manager

◎ *Script:* 📄 lifecycle_manager.py

◎ *Objective:*

- Create a central node that manages the lifecycle of all system nodes.
- Coordinate state transitions in the proper order.
- Monitor health of all managed nodes.
 - ▷ Publishers configured and activated before subscribers.
 - ▷ Subscribers deactivated before publishers.
 - ▷ Periodic health checks to verify all nodes remain active.

The Lifecycle Manager ensures proper system operation by:

- ◎ Ensuring data producers (publishers) are ready before data consumers (subscribers).
- ◎ Providing coordinated startup and shutdown sequences.
- ◎ Monitoring system health and detecting nodes that leave the active state.
- ◎ Using a multi-threaded executor for efficient processing.

✧ Implementation Hint

- ⦿ Initialize as a standard ROS2 node (not a lifecycle node).

```
| class LifecycleManager(Node):
```

- ⦿ The `__init__` method should:
 - Create instances of all lifecycle nodes (publishers and subscribers).

```
| self._temp_publisher = TemperaturePublisher()  
| self._humid_publisher = HumidityPublisher()  
| self._alert_generator = AlertGenerator()
```

- Group nodes into separate lists for publishers and subscribers.

```
| self._publisher_nodes = [self._temp_publisher, self._humid_publisher]  
| self._subscriber_nodes = [self._alert_generator]  
| self._all_nodes = self._publisher_nodes + self._subscriber_nodes
```

- Create a health monitoring timer.

```
| self.health_check_timer = self.create_timer(  
|     10.0, # Check every 10 seconds  
|     self._check_nodes_health  
| )
```

✳ Implementation Hint

- ⦿ Health monitoring timer callback.

```
def _check_nodes_health(self):
    """Periodically check the health of all nodes."""
    try:
        for node in self._all_nodes:
            # Get current state
            state = node.get_current_state()

            if hasattr(state, 'id'):
                state_id = state.id
                if state_id != 3: # Not ACTIVE
                    self.get_logger().warn(f'Node {node.get_name()} is not in ACTIVE state
                    ↳ (state: {state_id})')
                    # Could implement recovery logic here
    except Exception as e:
        self.get_logger().error(f'Error checking node health: {str(e)}')
```

Demonstration

```
[INFO] [1713811245.123456] [lifecycle_manager]: Creating lifecycle nodes
[INFO] [1713811245.234567] [lifecycle_manager]: Lifecycle Manager initialized
[INFO] [1713811245.345678] [lifecycle_manager]: Configuring all nodes
[INFO] [1713811246.456789] [lifecycle_manager]: All nodes configured
[INFO] [1713811246.567890] [lifecycle_manager]: Activating publisher nodes
[INFO] [1713811247.678901] [lifecycle_manager]: Activating subscriber nodes
[INFO] [1713811248.789012] [lifecycle_manager]: All nodes activated
[INFO] [1713811249.890123] [lifecycle_manager]: System is running. Press Ctrl+C to stop.
[INFO] [1713811260.901234] [lifecycle_manager]: All nodes healthy
```

Grading Rubric

✱ Grading Rubric (1/2) 20 pts

⦿ Exercise 1: Multi-Threaded Publisher-Subscriber System (10 pts)

• Data Publisher (3 pts)

- ▷ 2 pts: Correctly implement three publishers on different topics
- ▷ Timestamp publisher on `t data/timestamp`
- ▷ Random float publisher on `t data/random`
- ▷ Counter publisher on `t data/counter` with occasional value skipping
- ▷ 1 pt: Publish at 20 Hz with proper logging of skipped values

• Subscribers (7 pts)

- ▷ Moving Average Subscriber (2 pts)
- ▷ 1 pt: Implement sliding window of 10 values
- ▷ 1 pt: Log average only on initial window fill and significant changes
- ▷ Message Rate Subscriber (2 pts)
- ▷ 1 pt: Calculate and report publication rate every second
- ▷ 1 pt: Detect and log rate deviations from 20 Hz
- ▷ Counter Monitor Subscriber (3 pts)
- ▷ 1 pt: Track first received counter value
- ▷ 1 pt: Detect and log counter value discontinuities
- ▷ 1 pt: Implement robust error handling and logging

✧ Grading Rubric (2/2) 20 pts

⦿ Exercise 2: Lifecycle Management System (10 pts)

- Lifecycle Publishers (4 pts)

- ▷ Temperature Publisher (2 pts)

- ▷ 1 pt: Implement complete lifecycle management (configure, activate, deactivate, cleanup)

- ▷ 1 pt: Publish temperature within specified range with correct message details

- ▷ Humidity Publisher (2 pts)

- ▷ 1 pt: Implement complete lifecycle management (configure, activate, deactivate, cleanup)

- ▷ 1 pt: Publish humidity within specified range with correct message details

- Alert Generator (3 pts)

- ▷ 1 pt: Subscribe to temperature and humidity topics

- ▷ 1 pt: Implement thresholds for alerts (temperature and humidity)

- ▷ 1 pt: Only process data when node is in active state

- Lifecycle Manager (3 pts)

- ▷ 1 pt: Coordinate state transitions for all nodes

- ▷ 1 pt: Ensure proper order of node activation (publishers before subscribers)

- ▷ 1 pt: Implement periodic health checks for managed nodes

✱ Deductions

- ⦿ Late submission: *As per syllabus guidelines.*
- ⦿ Use of AI-generated code: Assignment grade of *zero*.
- ⦿ Code sharing/plagiarism: Assignment grade of *zero*.
- ⦿ Non-private GitHub repository: Assignment grade of *zero*.
- ⦿ Classes lack adequate documentation (missing docstrings) or insufficient inline comments: *-2 pts.*
- ⦿ If getters and setters are used in your program, ensure you do so with the `@property` decorator. If any other approaches are used, you will incur a penalty of *-2 pts.*
- ⦿ If even a single attribute is not prefixed with an underscore: *-2 pts.*