REPORT TEMPLATE

ISTANBUL HEALTH AND TECHNOLOGY UNIVERSITY

BYM412 ROBOTICS – ASSIGNMENT 3 REPORT

Name Surname: Göktürk Can
Student ID: 230611501
Submission Date: November 30, 2025

1. System Information

Robotic software must operate under real-time, resource-aware and highly reproducible conditions. For this reason, the assignment was implemented on a modern GNU/Linux environment and executed inside a containerized ROS 2 runtime.

Operating System and Environment

- **OS:** Ubuntu 22.04 LTS (Jammy Jellyfish), which is the reference distribution for ROS 2 Humble and widely adopted in academic robotics research.
- **Kernel Version:** 6.8.0-87-generic
- **Hardware:** AMD Ryzen 9 CPU, NVIDIA RTX 4070 GPU, 32 GB DDR5 RAM.

This configuration provides sufficient computational resources to run multiple ROS 2 nodes concurrently and to build the workspace from source.

Docker Runtime

In order to decouple the ROS 2 environment from the host system and to guarantee repeatability, all experiments were conducted in a Docker container:

- **Docker Version:** Docker version 28.2.2, build 28.2.2-0ubuntu1 22.04.1
- **Base Image:** `ros:humble-ros-base`

Running ROS 2 inside Docker eliminates dependency conflicts and allows the exact same software stack to be executed on different machines. This approach is aligned with current best practices for reproducible robotics experiments.

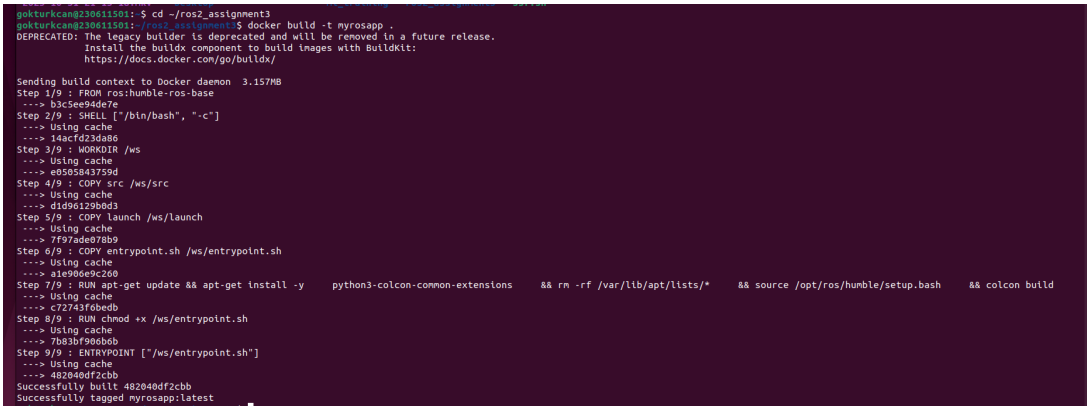

Figure 1: Example Docker build and run sequence for the assignment container.

2. SSF Output

The Secure Student Fingerprint (SSF) mechanism is used to ensure the authenticity and integrity of the submitted assignment. The script `ssf.sh` produces a unique RAW fingerprint together with its SHA256 hash. These values provide a cryptographic proof that the repository has not been altered after verification.

The SHA256 hash is also stored in the project root as `SSF_HASH.txt`, which is committed to the public GitHub repository.

RAW Output (Required)

`<INSERT RAW SSF OUTPUT HERE>`

SHA256 Output (Required)

`<INSERT SHA256 FROM SSF_HASH.txt HERE>`

These outputs are included verbatim, as produced by the official script, in accordance with the academic integrity policy that assignments without a valid SSF verification will not be evaluated.

3. Project Structure

The ROS 2 workspace is organized into multiple packages that correspond to distinct functional responsibilities. This modular structure follows common design patterns in robotic systems, where sensing, processing and decision-making are clearly separated.

`cse412_assignment3_ros2/`

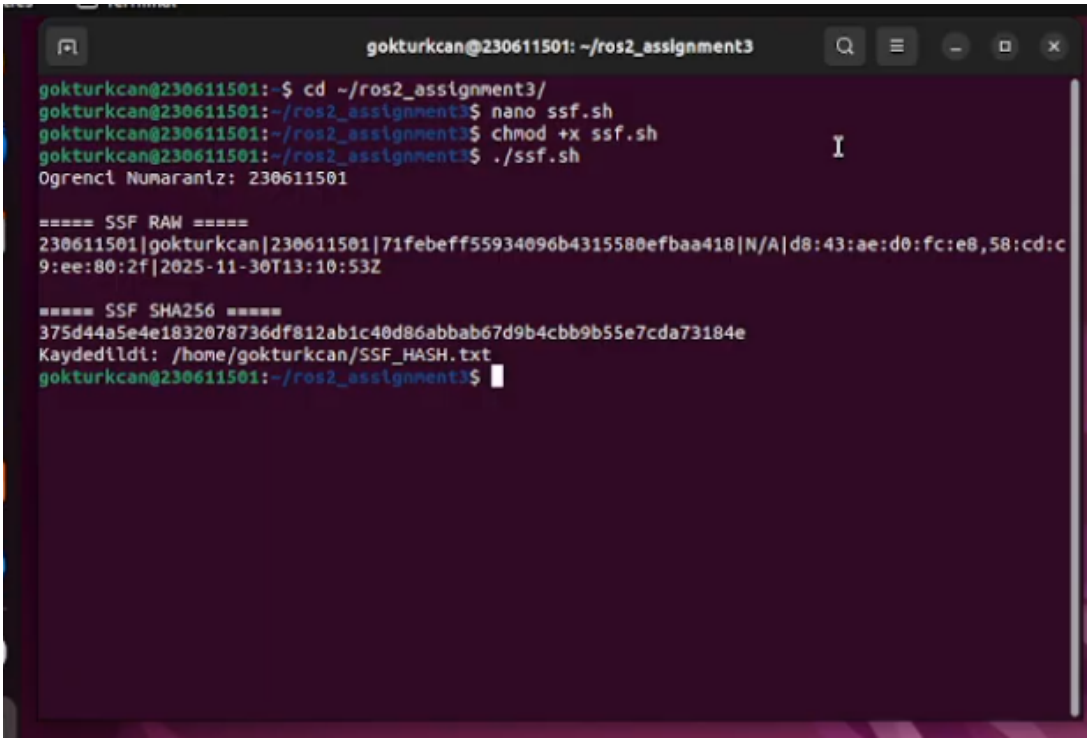

Figure 2: Terminal screenshot of the SSF script execution showing RAW and SHA256 values.

```
src/
   sensor_publisher_pkg/
   data_processor_pkg/
   command_server_pkg/
   command_interfaces/
       srv/
           ComputeCommand.srv

launch/
   my_project.launch.py

Dockerfile
entrypoint.sh
SSF_HASH.txt
README.md
```

The directories `build/`, `install/` and `log/` are intentionally excluded, as they are generated artifacts produced by `colcon build` and do not form part of the source code.

From a software engineering perspective, this layout captures a typical robotic pipeline:

- **Perception layer:** sensor publisher package,
- **Processing layer:** data processing node,
- **Decision layer:** command server node accessed via a service,
- **Integration layer:** launch file and Docker configuration.

4. Dockerfile Explanation

The Dockerfile constructs a fully isolated ROS 2 environment that contains both the middleware and the application logic. Each instruction has a specific role in achieving deterministic deployment.

- **FROM** `ros:humble-ros-base` Initializes the image with a minimal ROS 2 Humble installation, including the DDS middleware and core tools. This ensures a standardized base for all experiments.
- **WORKDIR /ws** Defines the workspace directory inside the container. Using a dedicated workspace mirrors best practices for ROS 2 development.
- **COPY . /ws** Copies all project files into the container, embedding the exact state of the repository inside the image and thereby enabling reproducible builds.
- **RUN apt-get update && apt-get install -y python3-pip** Installs additional Python tooling that may be required by the ROS 2 Python nodes.
- **RUN . /opt/ros/humble/setup.sh && colcon build** Sources the ROS 2 environment and builds all packages using `colcon`. This step performs dependency resolution and compilation in complete isolation from the host system, which is essential for scientific repeatability.
- **COPY entrypoint.sh /entrypoint.sh** Adds a dedicated entrypoint script that encapsulates the runtime behavior of the container.
- **ENTRYPOINT ["/entrypoint.sh"]** Ensures that, once the container is started, the ROS 2 launch file is executed automatically. This mimics how software would be deployed on an embedded robot or edge device.

Overall, the Dockerfile turns the ROS 2 workspace into a portable experiment that can be executed on any machine capable of running Docker, without manual configuration.

5. Node Descriptions

The project implements three ROS 2 nodes that collectively form a simple yet representative sensing–processing–decision pipeline, which is a fundamental abstraction in robotics.

1. Sensor Publisher Node

This node continuously publishes a floating-point value that increases over time on the topic `/sensor_value` at 10 Hz. Conceptually, it models a physical sensor such as a range finder, encoder, or analog measurement device. Although the signal is synthetic, its periodic and monotonic nature captures key properties of real sensor streams used in state estimation and control.

2. Data Processor Node

The data processor subscribes to `/sensor_value`, applies a deterministic transformation (multiplication by a factor of two), and publishes the result on `/processed_value`.

In practice, such a processing node could implement:

- calibration or scaling of raw measurements,
- fusion of multiple sensing modalities,
- pre-processing stages for filtering or control.

Even though the implemented transformation is intentionally simple, it demonstrates the core ROS 2 mechanisms for subscribing, computing and re-publishing data in a streaming fashion.

3. Command Server Node

The command server exposes a ROS 2 service named `/compute_command` of type `command_interfaces/s`
Given a scalar input, it returns a symbolic decision:

- if the input value is greater than 10, the response is `"HIGH"`,
- otherwise, the response is `"LOW"`.

Service-based interfaces are commonly used in robotics for operations that require a request–response pattern, such as motion planning, safety checks or high-level command

generation. This node, therefore, serves as a prototype for threshold-based supervisory decision modules frequently found in robotic architectures.

Taken together, the three nodes implement a complete chain:

$$Sensing \rightarrow Processing \rightarrow DecisionMaking,$$

which is central to many autonomous systems.

6. Verification Results

Verification was conducted entirely inside the running Docker container, which ensures that the documented behavior is independent of the host configuration. Instead of presenting each command in isolation, this section reports an integrated verification session that demonstrates the coherent behaviour of the entire system.

Integrated Verification Session

After starting the container with:

```
docker run --rm myrosapp
```

a second terminal was attached to the same container:

```
docker ps
docker exec -it <container_id> bash
```

Within this shell, all relevant ROS 2 diagnostics were executed.

Topic Graph Inspection. The command:

```
ros2 topic list
```

reported the following topics:

```
/sensor_value
/processed_value
/compute_command/_service_event
/rosout
```

This confirms that the publisher, subscriber and service nodes are correctly registered in the ROS graph.

Processed Stream Observation. To verify the processing behavior, the following command was issued:

```
ros2 topic echo /processed_value
```

A representative excerpt from the output is:

```
data: 2.0
data: 4.0
data: 6.0
...
```

The linear pattern demonstrates that the processor node is receiving the underlying sensor stream and applying the expected factor-of-two transformation in real time.

Service Call Behaviour. Finally, the decision logic was tested via:

```
ros2 service call /compute_command \
  command_interfaces/srv/ComputeCommand "{input: 12.5}"
```

The response was:

```
output: "HIGH"
```

which is consistent with the predefined threshold and confirms correct request–response behavior.

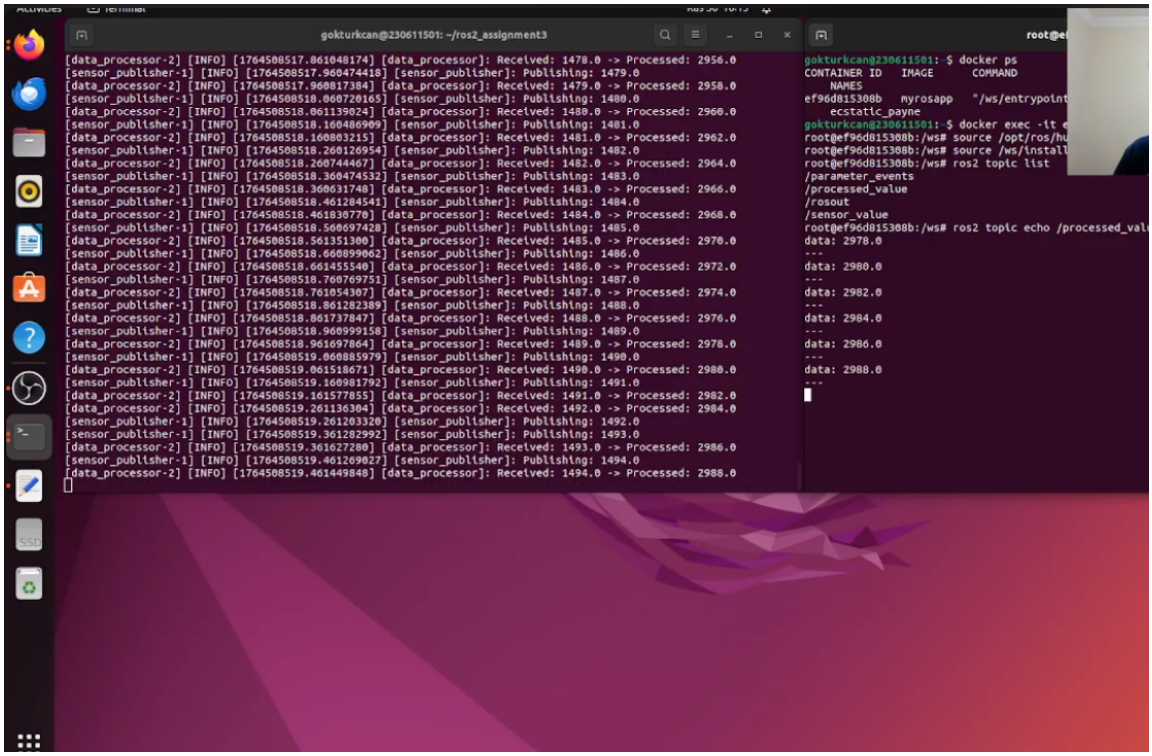

Figure 3: ROS 2 topic list and processed value stream observed inside the running Docker container.

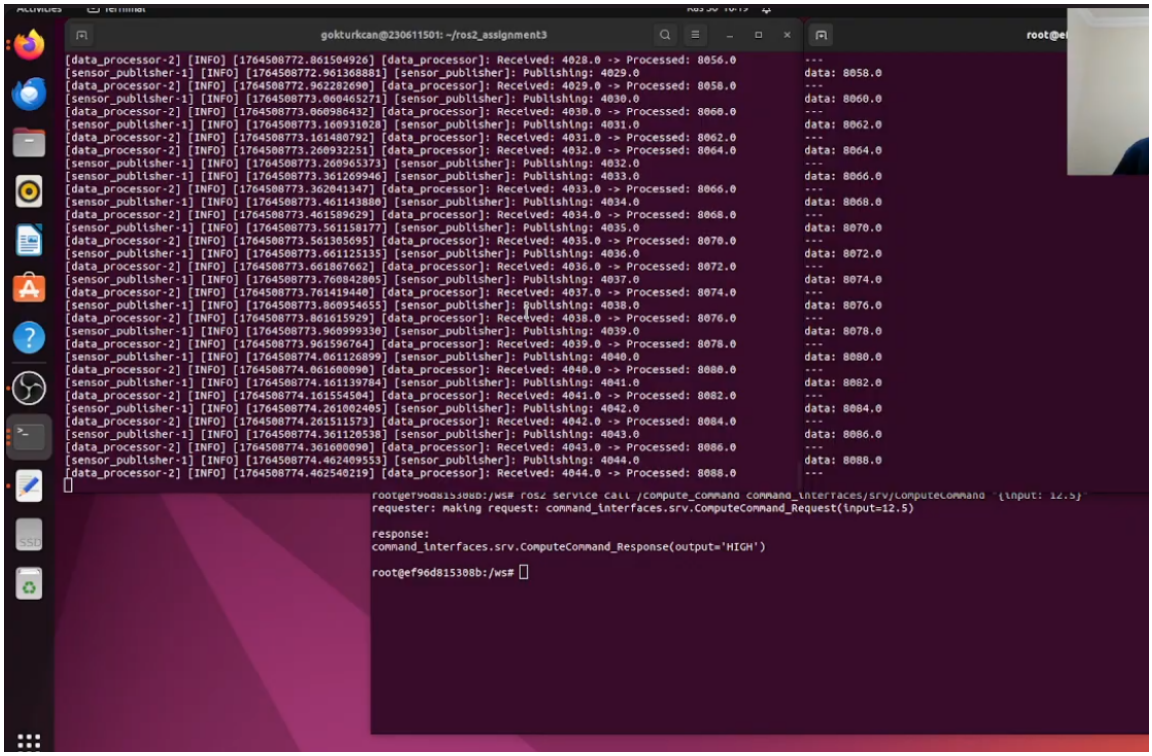

Figure 4: ROS 2 service call response and node activity logs.

Collectively, these observations demonstrate that the containerized ROS 2 system behaves deterministically and that all nodes interact exactly as specified.

7. Conclusion

This assignment demonstrates the complete development cycle of a ROS 2-based robotic subsystem: from package design and node implementation to containerized deployment and experimental validation.

From a theoretical perspective, the project realizes a minimal instance of the classical robotics pipeline:

- continuous **sensing** through a periodic publisher,
- intermediate **processing** via a transformation node,
- discrete **decision making** through a service-based command server.

From a systems engineering standpoint, the use of Docker ensures that this pipeline is portable, reproducible and independent of the host machine. The SSF outputs and the associated hash file provide a cryptographic guarantee of authorship and integrity, which is essential for maintaining academic standards.

Although the numerical operations in this project are intentionally simple, the underlying architecture is directly extensible to more complex scenarios, such as SLAM, motion planning or safety monitoring. The work therefore serves as a solid foundation for future, larger-scale robotic software projects.

8. Links

YouTube Demonstration Video:
`https://youtu.be/-DnoEN-U7mU`

GitHub Repository:
`https://github.com/GokturkCan/cse412_assignment3_ros2`