DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Università degli Studi di Padova – Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica

Relazione per la prova finale

Design and optimization of the software system for a self-driving boat

Relatore: Prof. Damiano Varagnolo

Laureando : Pietro Trabuio
Matricola n.: 2066686

Padova, 12/03/2026

Contents

Contents (ii)

1 Introduction

Introduction

The focus of my work was to integrate the software needed to run, communicate, and record data from the hardware used in the Nautilus self-driving project.

In particular the main areas of work for this thesis where:

- design of the ROS 2 module responsible for capturing and sharing data from the a stereo camera couple. It also needed to generate additional dynamic modules via configuration files
- integration of the Docker image for ROS 2
- design and implementation of a web server for viewing video streams

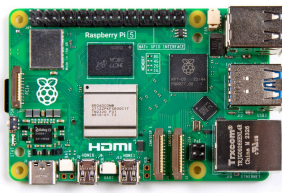
This presentation will also cover the Chioggia filed test we conducted with the Nautilus group.

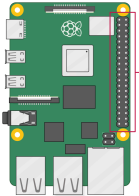
2 Hardware

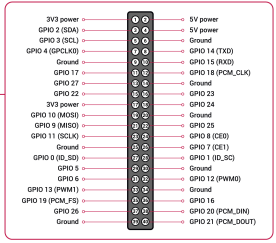
Raspberry Pi 5

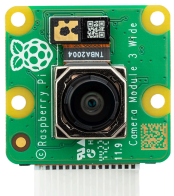
The choice of the Raspberry Pi 5 was dictated by its connectivity, in the form of the GPIO pins and camera connectors, form factor and power consumption.

Especially useful where the integrate camera connectors that allowed us to use well supported Camera module 3, avoiding potential compatibility risks with generic cameras.









Stero Camera Box

The box was designed to suport two cameras for stereo vision. It houses the raspberry and the cameras with the needed connections. It can be also made water resistant for submerision.

Boat and structure

We used both the BlueBoat for tests and for ease of deployment a custom structure mounted on a SUB board.

3 Software

ROS 2 project structure

The ROS 2 project structure is organized in a way that allows to easily integrate new modules and nodes. This comes naturally from the modular nature of ROS 2.

Right now in the project there are three main modules:

- the camera module
- the pump module
- the debug server.

```
ros2_ws/ # ROS2 workspace folder structure
└── src/
    ├── camera_module/       # camera module node folder
    │   ├── camera_module/   # camera node code
    │   ├── config/          # camera module config. files
    │   ├── launch/          # camera module launch files
    │   ├── resource/
    │   └── test/
    ├── debug_server/        # debug server node folder
    │   ├── debug_server/    # debug server code
    │   ├── launch/          # debug server launch files
    │   ├── resource/
    │   └── test/
    ├── node_pump/           # pump node folder
    │   ├── node_pump/       # pump node code
    │   ├── config/          # pump module config. files
    │   ├── launch/          # pump module launch files
    │   ├── resource/
    │   └── test/
    └── custom_pump_msgs/    # custom pump messages folder
        └── msg/             # custom message definitions
```

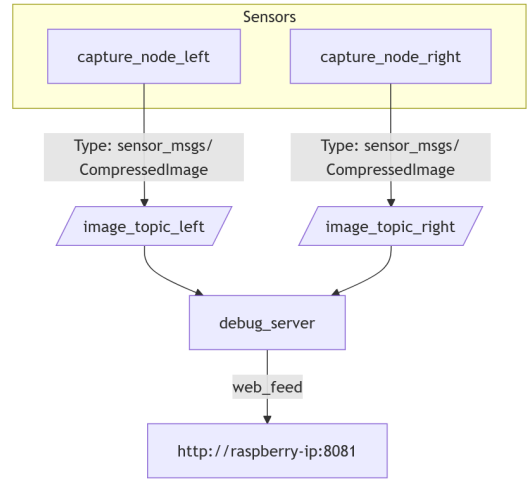
Camera node structure

The architecure of a single stereo module is made such in a way that can be replicated on multiple, for example, view directions.

Each module is contained in its own topic, so for example:

`front|back|.../camera_left|right`

It can also be seen the debug server that automatically and periodically searches for new topics and serves them on the local webserver.



Camera node code configuration

```
camera_module/
├── camera_module/
│   ├── __init__.py
│   └── capture_node.py
├── config/
│   └── stereo_camera.yaml
├── launch/
│   ├── stereo_camera_back.launch.py
│   ├── stereo_camera_configurable.launch.py
│   ├── stereo_camera_down.launch.py
│   └── stereo_camera_front.launch.py
├── resource/
├── test/
├── LICENSE
├── package.xml
├── setup.cfg
└── setup.py
```

```yaml
/**:
  ros__parameters:
    # Camera device settings
    camera_index: 0
    topic_name: 'image'

    # Image resolution
    width: 640
    height: 480

    # Frame rate
    fps: 30

    quality: 80

    use_compression: true

    # Publishing settings
    publish_rate: 30.0
    frame_id: "camera_frame"

    # System identification
    system_position: "front"
    camera_side: "left"
```

Docker environment and launch logic

To facilitate the deployment process, ROS 2 was installed and configured inside a Docker container. This allows for quick and consistent deployment across multiple systems with a single command.

So to deploy the system is sufficient to run a single docker command that will fetch, download and start the pre configured ROS 2 image.

For exact purpose ROS 2 developers created a custom image we based our system.

To simplify the process even further some shell scripts were created to automate the launch process, allowing to start the system with a single command.

Shell script for Docker

sh

```
# connect.sh
#!/bin/bash

CONTAINER_ID=$(docker ps -q --filter "name=nautilus-ros2*" | head -n 1)

if [ -n "$CONTAINER_ID" ]; then # connect to an alredy running instance
    echo "Connecting to running instance.\n"
    docker exec -it "$CONTAINER_ID" /bin/bash
else # build or start the container
    echo "Starting new instance.\n"
    docker compose run --rm raspberry
fi
```

Shell script to start the camera node

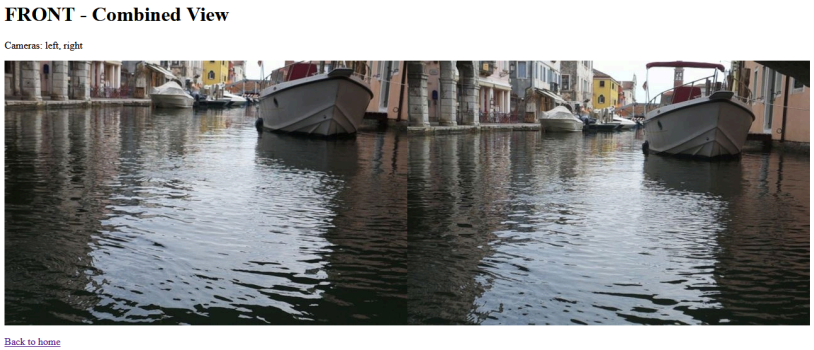
```sh
# launch-back-camera.sh
#!/bin/bash

if [ -f /.dockerenv ]; then
    cd /home/ubuntu/nautilus-ros2/ros2_ws
    source ./install/setup.bash
    ros2 launch camera_module stereo_camera_back.launch.py
else
    echo "Not running in Docker environment, execute 'connect.sh' to run the container."
    exit 1
fi
```

Debug server

The debug server is used to see the live feed of the camera modules for debugging and monitoring purposes.

It automatically searches for new camera topics and serves them on a local webserver.



4 On field experience

Chioggia canals

The system was tested on the Chioggia canals that revealed a challenging environment for both hardware and software.

System setup and startup

The deployed system was composed of two stereo camera module independent from each others.

To connect to both we used a phone as access point and logged in with ssh.

Here we started the ROS 2 node on both and a single debug server instance to monitor the cameras.

For recording the camera feed we started and saved different ROSBags that we used for testing later.

Problems and future improvements

One big problem was the overheating from the sun. The boxes being black and the limited cooling capacity of the raspberry heat-sink caused a stalling of the recording in different occasions. This could be solved using a bigger heat-sink or even implement some water-cooling solutions.

Also the ROSBags where saved on the same SD card that run the OS where the storage capacity is limited. A solution could be using an external USB storage device.

On the software side starting multiple ssh session for each module is impractical so in the future it should be implemented a centralized solution to start the modules and the recordings.

5 Conclusions

Conclusions

Working on this project was a great experience that allowed me to apply and expand my knowledge in software development, robotics and system integration.

The project allowed me to make experience managing a complex software project and working with a team of people.

The field tests conducted in the canals of Chioggia provided a practical validation of the system, highlighting both its strengths and areas for improvement.

Overall, this work contributes to the still early development of the Nautilus autonomous marine vehicle and lays a solid base for future expansion and development.